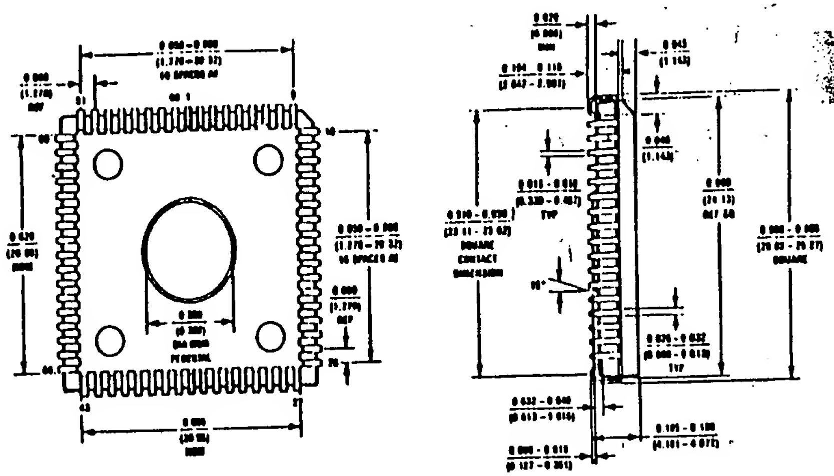
EXHIBIT A

Physical Dimensions inches (Millimeters)



Plastic Chip Carrier (V)
Order Number NS32FX16V-15 or NS32FX16V-20 or NS32FX16V-25
NS Package Number V68A

LIFE SUPPORT POLICY

National Semiconductor

ADVANCED INFORMATION
January 1990

NS32FX16-15/NS32FX16-20/NS32FX16-25 High Performance FAX Processor

General Description

The NS32FX16 is a high-performance 32-bit Embedded System Processor ® that is optimized for Group 2 and Group 3 Facsimile applications, Data Modems, Voice Mail systems and Laser Printers. It performs all the computations and control functions required for a stand-alone FAX system, a PC add-in FAX/Data modem card or a Laser/FAX system. The NS32FX16 can execute, in real time, V.29 (9600 bps and 7200 bps), V.27 (4800 bps and 2400 bps), and V.21. The NS32FX16 incorporates four main modules: the NS32CG16 compatible CPU Core, a 384-Byte Memory Array, a FAX Accelerator Module and a Bus Interface Unit.

The CPU Core incorporates a full 32-bit ALU and 32-bit internal data bus. This processor also supports a 16-Mbyte linear address space, a 16-bit external data bus and an 8 byte prefetch queue.

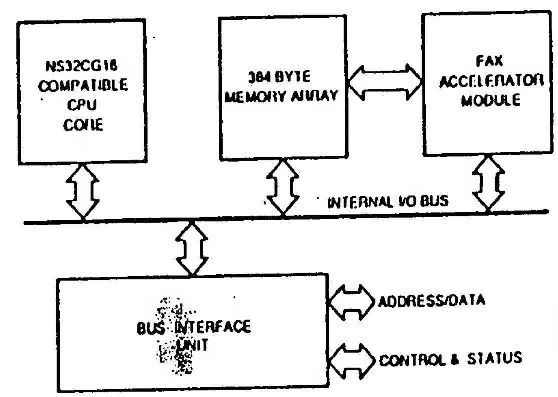
The FAX Accelerator Module (FAM) executes vector operations on complex variables and is optimized for Modem applications. It is designed to enhance performance on modem Digital Signal Processing (DSP) primitives while preserving the CPU core's structure and programming model. The vector operations can be used to efficiently implement FIR filters and other DSP primitives.

The FAM is attached to the CPU core via the Internal I/O Bus. It is treated as a memory mapped I/O device, occupying a reserved memory space. The CPU controls this module via a set of memory mapped registers. The module reduces the load of the main processor by fetching operands using its own address generator. In order to save bus bandwidth, the FAM stores the coefficients of the various filters in an internal 384-byte Memory Array. The 384-byte Memory Array is a shared resource and is usable by both the FAM and the CPU core.

Besides the highly efficient architecture and the addition of the FAM, the NS32FX16 supports all the NS32CG16 instructions including graphic enhancements like BitBLT (Bit-aligned BLock Transfer) operations and other special graphic instructions. These graphic enhancements can be used to support Postscript™ applications such as printers and Laser/FAX machines.

The microprocessor is packed in a 68-pin Plastic Leaded Chip Carrier (PLCC) package.

Block Diagram



Features

- 32-bit architecture and implementation
- 16-Mbyte linear addressing space
- On-chip FAX Accelerator Module for DSP support
- Special support for graphics applications
  - 18 graphics instructions
  - Efficient fonts & pattern handling
  - Interface to an external BitBLT processing units for fast color BitBLT operations
- 384-byte on-chip RAM array
- Operating frequency 15, 20, and 25 MHz
- Binary compatible with the Series 32000 ® family
- Floating-point support via the NS32081 or the NS32381
- Power save mode
- Double-metal CMOS technology
- 68 pin PLCC package
- On-chip clock generator

Product Introduction

The NS32FX16 is a high speed CMOS microprocessor in National's Embedded System Processor family. It includes two main execution units: the NS32CG16 compatible CPU Core and the FAX Accelerator Module. The CPU Core is designed for general purpose computations and system control functions. The FAX Accelerator Module is tuned to perform the DSP primitives needed in Voice Band Modems. The NS32FX16 also incorporates a 384-byte Memory Array as a shared resource for both the CPU core and the FAX Accelerator Module. These features make the NS32FX16 an optimal solution for applications in which both general purpose and DSP computations are needed. Such applications include FAX Modems, Voice Compression, and Voice Mail systems.

The NS32FX16 is software-compatible with all other CPUs in the family. The device incorporates all of National's Embedded System Processor advanced architectural features, with the exception of the virtual memory capability.

Brief descriptions of the NS32FX16 features that are shared with other members of the family are provided below:

Powerful Addressing Modes. Nine addressing modes available to all instructions are included to access data structures efficiently.

Data Types. The architecture provides for numerous data types, such as byte, word, doubleword, and BCD, which may be arranged into a wide variety of data structures.

Symmetric Instruction Set. While avoiding special case instructions that compilers can't use, National's Embedded System Processor family incorporates powerful instructions for control operations, such as array indexing and external procedure calls, which save considerable space and time for compiled code.

Memory-to-Memory Operations. National's Embedded System Processor CPUs represent two-address machines. This means that each operand can be referenced by any one of the addressing modes provided.

This powerful memory-to-memory architecture permits memory locations to be treated as registers for all useful operations. This is important for temporary operands as well as for context switching.

Large, Uniform Addressing. The NS32FX16 has 32-bit address pointers that can address up to 16 Mbytes of external memory without any segmentation; this addressing scheme provides flexible memory management without added-on expense.

Modular Software Support. Any software package for National's Embedded System Processor family can be developed independent of all other packages, without regard to individual addressing. In addition, ROM code is totally relocatable and easy to access, which allows a significant reduction in hardware and software costs.

Software Processor Concept. National's Embedded System Processor architecture allows expansions of the instruction set that can be executed by Floating Point slave processors, acting as extensions to the CPU. Current Floating Point slave processors of the Embedded System Processor family are the NS32081 and the NS32381.

To summarize, the architectural features cited above provide three primary performance advantages and characteristics:

- High-Level Language Support
- Easy Future Growth Path
- Application Flexibility

Table of Contents

B.4 FAX ACCELERATOR MODULE PERFORMANCE

B.4.1 Assumptions

The FAM instruction execution starts with the CPU core writing to the CTTL register. The execution time is counted from T3 of this transaction until all the results are ready, either in the Accumulator or in the Coefficient array.

It is assumed that there are:

- No wait states in FAM initiated read cycles
- No external HOLD request during the FAM operation for VCMAD, VCMUL and VCMAC instructions.

B.4.2 Definitions

N Number of elements in complex vector

TFAM Number of clock cycles to execute the instruction

B.4.3 FAM Performance in Clock Cycles

FAM instruction	TFAM
VCMAD	9 + (N * 8)
VCMUL	9 + (N * 8)
VCMAC	6 + (N * 8)
VCMAG	5 + (N * 8)

endix B: NS32FX16 Instruction Timing (Continued)

6 NS32081 Floating-Point Execution Times

following section gives execution timing information for Floating-Point Instructions. Some additional timing itions are used, as given below:

The floating-point operation length.
Standard Floating (32 bits): f = 1
Long Floating (64 bits): f = 2

The time required to transfer 32 bits of a floating-point value to or from the NS32081 Floating-Point unit.
TI = 4 always

The time required to transfer an integer value to or from the NS32081 Floating-Point Unit.
Byte: TI = 2
Word: TI = 2
Double-word: TI = 4

TABLE B-5. Floating-Point Instruction Execution Times

EMONIC	CASE	TEA	TOPD	TOPI	TI	TI	TCY
vf	<MM>	2	2f	-	-	2f	23
	<RR>	-	-	-	-	-	27
	<MR>	1	f	-	-	f	23
	<RM>	-	f	-	-	f	27
I, SUBf	<MM>	2	3f	-	-	3f	70
	<RR>	-	-	-	-	-	74
	<MR>	1	f	-	-	f	70
	<RM>	1	2f	-	-	2f	70
f	<MM>	2	3f	-	-	3f	30+14f
	<RR>	-	-	-	-	-	34+14f
	<MR>	1	f	-	-	f	30+14f
	<RM>	1	2f	-	-	2f	30+14f
	<MM>	2	3f	-	-	3f	55+30f
	<RR>	-	-	-	-	-	59+30f
	<MR>	1	f	-	-	f	55+30f
	<RM>	1	2f	-	-	2f	55+30f
f, NEGf	<MM>	1	2f	-	-	2f	20
	<RR>	-	-	-	-	-	24
	<MR>	1	f	-	-	f	20
	<RM>	-	f	-	-	f	24
f	<MM>	1	2f	-	-	2f	45
	<RR>	-	-	-	-	-	49
	<MR>	1	f	-	-	f	45
	<RM>	1	f	-	-	f	45
/LF	<MM>	1	3	-	-	3	23
	<RR>	-	-	-	-	-	27
	<MR>	1	2	-	-	2	23
	<RM>	-	1	-	-	1	27
/FL	<MM>	1	3	-	-	3	22
	<RR>	-	-	-	-	-	26
	<MR>	1	1	-	-	1	22
	<RM>	-	2	-	-	2	26
/f	<MM>	1	f	1	1	f	53
	<MR>	1	-	1	1	-	53
IND, TRUNCf	<MM>	1	f	1	1	f	53
ORf	<RM>	-	-	1	1	-	66
R	<M>	-	1	-	-	1	13
R	<M>	1	1	-	-	1	18

List of Illustrations (Continued)

List of Tables

1.0 Product Information

1.1 NS32FX16 SPECIAL FEATURES

In addition to the above National Embedded System Processor features, the NS32FX16 provides features that make the device extremely attractive for a wide range of applications where Digital Signal Processing, graphics support, low chip count, and low power consumption are required.

The most relevant of these features are the enhanced Digital Signal Processing performance, which makes the chip very attractive for usage in facsimile, and the graphics support capabilities, that can be used in applications such as printers, CRT terminals, and other varieties of display systems, where text and graphics are to be handled.

Graphics support is provided by eighteen instructions that allow operations such as BitBLT, data compression/expansion, fills, and line drawing, to be performed very efficiently. In addition, the device can be easily interfaced to an external BitBLT Processing Unit (BPU) for high BitBLT performance.

The NS32FX16 allows systems to be built with a relatively small amount of random logic. With the external DMA support, the bus can be highly optimized to allow simple interfacing to a large variety of RAMs and peripheral devices. All the relevant bus access signals and clock signals are generated on-chip. The cycle extension logic is also incorporated on-chip.

The device is fabricated in a low-power, double metal, CMOS technology. It also includes a power-save feature that allows the clock to be slowed down under software control, thus minimizing the power consumption. This feature can be used in those applications where power saving during periods of low performance demand is highly desirable.

The bus characteristics and the power save feature are described in the "Functional Description" section. A description of the FAX Accelerator Module is provided in Section 2.5. A general overview of BitBLT operations and a description of the graphics support instructions is provided in Section 2.4. Details on all the NS32FX16 instructions can be found in the NS32CG16 Printer/Display Processor Programmer's Reference Supplement and the related NS32CG16 supplement.

Below is a summary of the instructions that are directly applicable to graphics along with their intended use.

Instruction	Application
BBAND BBFOR BBOR BBXOR BBSTOD BITWT EXTBLT	The BitBLT group of instructions provide a method of quickly imaging characters, creating patterns, windowing and other block oriented effects.
MOVMP	Move Multiple Pattern is a very fast instruction for clearing memory and drawing patterns and lines.
TBITS	Test Bit String will measure the length of 1's or 0's in an image, supporting many data compression methods (RLL). TBITS may also be used to test for boundaries of images.
SBITS	Set Bit String is a very fast instruction for filling objects, outline characters and drawing horizontal lines. The TBITS and SBITS instructions support the CCITT standard for compression/decompression algorithms used by Group 3 and Group 4 facsimile machines.
SBITPS	Set Bit Perpendicular String is a very fast instruction for drawing vertical, horizontal and 45° lines. In printing applications SBITS and SBITPS may be used to express portrait and landscape respectively from the same compressed font data. The size of the character may be scaled as it is drawn.
SBIT CBIT TBIT IBIT	The Bit Group of instructions enable single pixels anywhere in memory to be set, cleared, tested or inverted.
INDEX	The INDEX instruction combines a multiply-add sequence into a single instruction. This provides a fast translation of an X-Y address to a pixel relative address.

2.0 Architectural Description

2.1 REGISTER SET

The NS32FX16 CPU core has 17 internal registers grouped according to function as follows: 8 general purpose, 7 address, 1 processor status and 1 configuration. *Figure 2-1* shows the NS32FX16 internal registers of the CPU core.

Besides the CPU core registers, the NS32FX16 also has 6 registers, FAX Accelerators and 384 byte RAM which can be accessed as memory-mapped I/O. Refer to section 2.5 for more details.

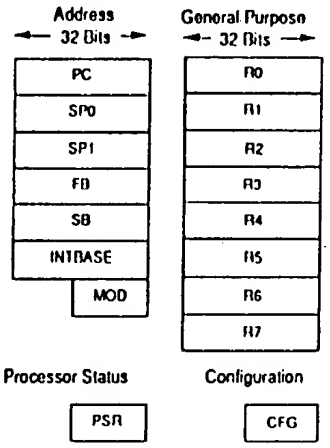


FIGURE 2-1. NS32FX16 Internal CPU Core Registers

TABLE B-4. Instruction Timing Parameters

MNEMONIC	TEA	TOPB	TOPW	TOPD	TOPI	TCY	L	NOTES
INSi	2	-	-	2	1	29-39	-	field in memory
INSi	1	-	-	-	1	28-96	-	field in register
INSSi	2	-	-	2	1	39-49	-	
JSR	1	-	-	1	1	5*15	-	
JUMP	1	-	-	-	-	2*6	-	
LPRi	1	-	-	-	1	19-33	-	
LSRi	2	1	-	-	2	14-45	-	
MEI	2	-	-	-	4	23	16	
MOD	2	-	-	-	3	54-73	16	
MOVi	2/1/0	-	-	-	2/1/0	1/3/3	-	<xM>/<MR>/<RR>
MOVMi	2	-	-	-	2*n	3*n+20	-	n = # of elements in block
MOVQi	1/0	-	-	-	1/0	2/3	-	<M><R>
MOVSi	-	-	-	-	-	-	-	See graphics instructions
MOVSi	-	-	-	-	2*n	24*n+54	-	B, W and/or M option in effect
MOVST	-	n	-	-	2*n	27*n+54	-	Translated
MOVXBD	2	1	-	1	-	6	-	
MOVXBW	2	1	1	-	-	6	-	
MOVXWD	2	-	1	1	-	6	-	
MOVZBD	2	1	-	1	-	5	-	
MOVZBW	2	1	1	-	-	5	-	
MOVZWD	2	-	1	1	-	5	-	
MULi	2	-	-	-	3	15	16	
NEGi	2	-	-	-	2	5	16	
NOP	-	-	-	-	-	3	-	
NOTi	2	-	-	-	2	5	-	
ORi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
QUOi	2	-	-	-	3	49-55	16	
REM	2	-	-	-	3	57-62	16	
RESTORE	-	-	-	n	-	5*n+12	-	n=# of general registers restored
RET	-	-	-	1	-	2*8	-	
RETi	-	1	3	3	-	39*45	-	
RETT	-	-	2	2	-	35*41	-	
ROTi	2	1	-	-	2	14-45	-	
RXP	-	-	1	2	-	2*6	-	
Scondi	1	-	-	-	-	1	9/10	False/True
SAVE	-	-	-	n	-	4*n+13	-	n=# of general registers saved
SBITi	2/1	2/0	-	-	1	15/7	-	<xM>/<xR>
SBITIi	2/1	2/0	-	-	1	15/7	-	<xM>/<xR>
SETCFG	-	-	-	-	-	15	-	
SKPSi	-	-	-	-	n	27*n+51	-	n=# of elements, not translated
SKPST	-	n	-	-	n	30*n+51	-	Translated
SPRi	1	-	-	-	1	21-27	-	
SUBi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
SUBCi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<R>/<RR>
SUBPi	2	-	-	-	3	16/18	-	no carry/carry
SVC	-	-	4	3	-	40	-	
TBITi	2/1	1/0	-	-	1	14/4	-	<xM>/<xR>
WAIT	-	-	-	-	-	6-7	-	? - until an interrupt/reset
XORi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>

ppendix B: NS32FX16 Instruction Timing (Continued)

TABLE B-4. Instruction Timing Parameters

NEMONIC	TEA	TOPB	TOPW	TOPD	TOPI	TCY	L	NOTES
BSi	2	-	-	-	2	9/8	-	src <0/src> = 0
CBi	1	-	-	-	2	16/15%2(		<M>, no branch/branch
CBi	-	-	-	-	-	18/17%2		<R>, no branch/branch
DDi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
DDCi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
DDPi	2	-	-	-	3	16/18	-	no carry/carry
DDQi	1/0	-	-	-	2/0	6/4	-	<M>/<R>
DDR	2/1	-	-	1/0	-	2/3	-	<xM>/<xR>
DJ	1	-	-	-	1	6	-	<xM>/<MR>/<RR>
ND	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
SHi	2	1	-	-	2	14-45	-	
cond	-	-	-	-	-	7/6%10	-	no branch/branch
Ci	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
CPSRB	1	1	-	-	-	18%22	-	
CPSRW	1	-	1	-	-	30%34	-	
SPSRB	1	1	-	-	-	18%22	-	
SPSRW	1	-	1	-	-	30%34	-	
PT	-	-	4	3	-	40	-	
R	1	-	-	-	1	4%0	-	
SR	-	-	-	1	-	6%16	-	
ASEi	1	-	-	-	1	4%9	-	
BITi	2/1	2/0	-	-	1	15/7	-	<xM>/<xR>
BITi	2/1	2/0	-	-	1	15/7	-	<xM>/<xR>
HECKi	2	-	-	-	3	7/10/11	-	high/low/ok
MPi	2/1/0	-	-	-	2/1/0	3	-	<xM>/<MR>/<RR>
MPMi	2	-	-	-	2*n	9*n + 24	-	n = # of elements in block
MPQi	1/0	-	-	-	1/0	3	-	<M>/<R>
MPSi	-	-	-	-	2*n	35*n+53	-	n = # of elements, not Translated
MPST	-	n	-	-	2*n	38*n+53		Translated
C	2	-	-	-	2	7	-	
V	2	-	-	1	-	7	-	
XP	-	-	3	4	-	16%21	-	
XPD	1	-	3	3	-	13%18	-	
EU	2/1	-	-	-	5/1	38/31	16	<xM>/<xR>
IA	-	-	-	-	-	3%7	-	
IVI	2	-	-	-	3	58-68	16	
NTER	-	-	-	n + 1	-	4*n + 18	-	n = # of general registers
								saved
XIT	-	-	-	n + 1	-	5*n + 17	-	n = # of general registers restored
XTi	2	-	-	1	1	19-29	-	field in memory
XTi	2	-	-	-	1	17-51	-	field in register
XTSi	2	-	-	1	1	26-36	-	
FSi	2	2	-	-	1.	24-28	24	
LAG	-	-	0/4	0/3	-	6/44	-	no trap/trap
IITi	2/1	2/0	-	-	1	17/9	-	<xM>/<xR>
NDEXi	2	-	-	-	2	25	16	

2.0 Architectural Description (Continued)

2.1.1 General Purpose Registers

There are eight registers (R0-R7) used for satisfying the high speed general storage requirements, such as holding temporary variables and addresses. The general purpose registers are free for any use by the programmer. They are 32 bits in length. If a general purpose register is specified for an operand that is 8 or 16 bits long, only the low part of the register is used; the high part is not referenced or modified.

2.1.2 Address Registers

The seven address registers are used by the processor to implement specific address functions. Except for the MOD register that is 16 bits wide, all the others are 32 bits. A description of the address registers follows.

PC—Program Counter. The PC register is a pointer to the first byte of the instruction currently being executed. The PC is used to reference memory in the program section.

SP0, SP1—Stack Pointers. The SP0 register points to the lowest address of the last item stored on the INTERRUPT STACK. This stack is normally used only by the operating system. It is used primarily for storing temporary data, and holding return information for operating system subroutines and interrupt and trap service routines. The SP1 register points to the lowest address of the last item stored on the USER STACK. This stack is used by normal user programs to hold temporary data and subroutine return information.

When a reference is made to the selected Stack Pointer (see PSR S-bit), the terms 'SP Register' or 'SP' are used. SP refers to either SP0 or SP1, depending on the setting of the S bit in the PSR register. If the S bit in the PSR is 0, SP refers to SP0. If the S bit in the PSR is 1 then SP refers to SP1.

Stacks in the Series 32000 family grow downward in memory. A Push operation pre-decrements the Stack Pointer by the operand length. A Pop operation post-increments the Stack Pointer by the operand length.

FP—Frame Pointer. The FP register is used by a procedure to access parameters and local variables on the stack. The FP register is set up on procedure entry with the ENTER instruction and restored on procedure termination with the EXIT instruction.

The frame pointer holds the address in memory occupied by the old contents of the frame pointer.

SB—Static Base. The SB register points to the global variables of a software module. This register is used to support relocatable global variables for software modules. The SB register holds the lowest address in memory occupied by the global variables of a module.

INTBASE—Interrupt Base. The INTBASE register holds the address of the dispatch table for interrupts and traps (Section 3.7.1).

MOD—Module. The MOD register holds the address of the module descriptor of the currently executing software module. The MOD register is 16 bits long; therefore, the module table must be contained within the first 64 Kbytes of memory.

2.1.3 Processor Status Register

The Processor Status Register (PSR) holds status information for the microprocessor.

The PSR is sixteen bits long, divided into two eight-bit halves. The low order eight bits are accessible to all programs, but the high order eight bits are accessible only to programs executing in Supervisor Mode.

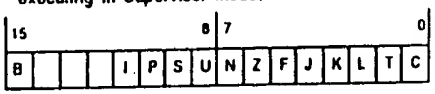


FIGURE 2-2. Processor Status Register (PSR)

- **C** The C bit indicates that a carry or borrow occurred after an addition or subtraction instruction. It can be used with the ADDC and SUBC instructions to perform multiple precision integer arithmetic calculations. It may have a setting of 0 (no carry or borrow) or 1 (carry or borrow).
- **T** The T bit causes program tracing. If this bit is set to 1, a TRC trap is executed after every instruction (Section 3.7.6).
- **L** The L bit is altered by comparison instructions. In a comparison instruction the L bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as unsigned integers. Otherwise, it is set to "0". In floating-point comparisons, this bit is always cleared.
- **K** Reserved for use by the CPU.
- **J** Reserved for use by the CPU.
- **F** The F bit is a general condition flag, which is altered by many instructions (e.g., integer arithmetic instructions use it to indicate overflow).
- **Z** The Z bit is altered by comparison instructions. In a comparison instruction, the Z bit is set to "1" if the second operand is equal to the first operand; otherwise it is set to "0".
- **N** The N bit is altered by comparison instructions. In a comparison instruction the N bit is set to "1" if the second operand is less than the first operand, when both operands are interpreted as signed integers. Otherwise, it is set to "0".
- **U** If the U bit is "1" no privileged instructions may be executed. If the U bit is "0" then all instructions may be executed. When U = 0 the processor is said to be in Supervisor Mode; when U = 1 the processor is said to be in User Mode. A User Mode program is restricted from executing certain instructions and accessing certain registers which could interfere with the operating system. For example, a User Mode program is prevented from changing the setting of the flag used to indicate its own privilege mode.

A Supervisor Mode program is assumed to be a trusted part of the operating system, hence it has no such restrictions.

S The S bit specifies whether the SP0 register or SP1 register is used as the Stack Pointer. The bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use the SP0 register) or 1 (use the SP1 register).

P The P bit prevents a TRC trap from occurring more than once for an instruction (Section 3.7.6). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I If I=1, then all interrupts will be accepted. If I=0, only the NMI interrupt is accepted. Trap enables are not affected by this bit.

B Reserved for use by the CPU. This bit is set to 1 during the execution of the EXTBLT instruction and causes the $\overline{BPU}$ signal to become active. Upon reset, B is set to zero and the $\overline{BPU}$ signal is set high.

Note 1: When an interrupt is acknowledged, the B, I, P, S and U bits are set to zero and the $\overline{BPU}$ signal is set high. A return from interrupt will restore the original values from the copy of the PSR register saved in the interrupt stack.

Note 2: If BitBLT (BB) instructions are executed in an interrupt routine, the PSR bits J and K must be cleared first.

2.1.4 Configuration Register

The Configuration Register (CFG) is 8 bits wide, of which 4 bits are implemented. The implemented bits enable various operating modes for the CPU, including execution of floating-point instructions, processing of exceptions and selection of clock scaling factor. CFG is programmed by the SETCFG instruction. The format of CFG is shown in *Figure 2-3*. The various control bits are described below.

FIGURE 2-3. Configuration Register (CFG)

I Interrupt vectoring. This bit controls whether maskable interrupts are handled in nonvectored (I = 0) or vectored (I = 1) mode. Refer to Section 3.7.3 for more information.

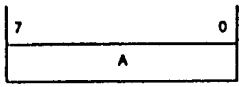
F Floating-point instruction set. This bit indicates whether a floating-point unit (FPU) is present to execute floating-point instructions. If this bit is 0 when the CPU executes a floating-point instruction, a Trap (UND) occurs. If this bit is 1, then the CPU transfers the instruction and any necessary operands to the FPU using the slave-processor protocol described in Section 3.8.1.

M Clock scaling. This bit is used in conjunction with the C bit to select the clock scaling factor.

C Clock scaling. Same as the M bit above. Refer to Section 3.2.1 on "Power Save Mode" for details.

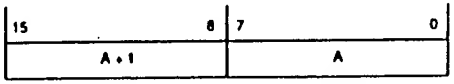
2.2 MEMORY ORGANIZATION

The NS32FX16's external address space is a uniform 16 Mbyte (24-bit address) linear address space. Memory locations are numbered sequentially starting at zero and ending at $2^{24}-1$. The number specifying a memory location is called an address. The contents of each memory location is a byte consisting of eight bits. Unless otherwise noted, diagrams in this document show data stored in memory with the lowest address on the right and the highest address on the left. Also, when data is shown vertically, the lowest address is at the top of the diagram and the highest address at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number zero, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.



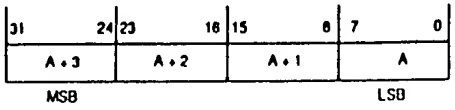
Byte at Address A

Two contiguous bytes are called a word. Except where noted, the least significant byte of a word is stored at the lower address, and the most significant byte of the word is stored at the next higher address. In memory, the address of a word is the address of its least significant byte, and a word may start at any address.



Word at Address

Two contiguous words are called a double-word. Except where noted, the least significant word of a double-word is stored at the lowest address and the most significant word of the double-word is stored at the address two higher. In memory, the address of a double-word is the address of its least significant byte, and a double-word may start at any address.



Double Word at Address A

Although memory is addressed as bytes, it is actually organized as words. Therefore, words and double-words that are aligned to start at even addresses (multiples of two) are accessed more quickly than words and double-words that are not so aligned.

B.3.3 Calculation of Total Execution Time (TEX)

TEX is obtained by performing the following steps:

1) Find the desired instruction in the table.
2) Calculate the values of TEA, TOPI, etc., using the numbers in the table and the equations given on the preceding page.
3) The result derived by adding together these values is the execution time (TEX) in clock cycles.

B.3.4 Notes on Table Use

Values in the TEA column (see Tables B-4 and B-5) indicate the number of effective addresses to be calculated. If the value in this column is less than the number of general operands in the instruction, this is because one or both operands are in registers and the instruction has an optimized form which eliminates TEA for such operands.

In the L column, multiply the entry by the operation length in bytes (1, 2 or 4).

In the TCY column, special notations sometimes appear:

n1-n2 means n1 minimum, n2 maximum.
n1%n2 means that the instruction flushes the instruction queue after n1 clock cycles and nonsequentially fetches the next instruction. The value n2, indicating the total number of clock cycles in internally executing the instruction (including n1), is not generally useful. The most accurate technique for determining such timing depends on the size and alignment of the basic instruction portion of the next instruction, plus index bytes. If this portion can be read in one memory cycle, then the execution time is n1+10 (including the memory cycle). If more memory cycles are required, the value is n1+5+4*m, where m is the number of memory cycles required.

In the Notes column, notations held within angle brackets <> indicate alternatives in the form of the instruction which affect the execution time. A table entry which is affected by the form of the instruction may have multiple values, separated by slashes, corresponding to the alternatives. The notations are:

<M>	Memory form
<R>	Register form
<MM>	Memory-to-Memory form
<IM>	Register-to-Memory form
<MR>	Memory-to-Register form
<RR>	Register-to-Register form
x	either Register to Memory

B.3.5 Example of Table Usage

Calculate TEX for the instruction: CMPW R0, TOS.

Operand A is in a register; Operand B is in memory. The table values must be used corresponding to the <xM> case as given in the Notes column (<xM> meaning "anything to memory").

Only the TEA, TOPI and TCY columns have values assigned for the CMPi instruction; therefore, they are the only ones that need to be calculated to find TEX. The blank columns are irrelevant to this instruction.

The TEA column contains 2 for the <xM> case. This means that effective address times have to be calculated for both operands. (For the <MR> case, the Register operand requires no TEA time; therefore, only the Memory operand TEA is necessary.) From the equations:

TEA (Register mode) = 2.
TEA (Top-of-Stack mode, read access class) = 2.
Total TEA = 2 + 2 = 4.

The TOPI column represents potential operand transfers to or from memory. For a Compare instruction, each operand is read once, for a total of two operand transfers

TOPi (Word, Register) = 0.
TOPi (Word, TOS) = TOPW = 3 (assuming aligned, no waits)
Total TOPI = 3

TCY is the time required for internal operation within the CPU. The TCY value for this case is 3.

TEX = TEA + TOPi + TCY = 4 + 3 + 3 = 10 machine cycles.

If the CPU is running at 15 MHz, then a machine cycle (clock cycle) is 66 ns. Therefore, this instruction takes 10 x 66 ns, or 660 ns to execute.

Appendix B: NS32FX16 Instruction Timing (Continued)

B.3.2 Definitions

TEA — The time required to calculate an operand's effective address. For a Register or Immediate operand, this includes the fetch of that operand.

TOPB — The time needed to read or write a memory byte.

TOPW — The time needed to read or write a memory word.

TOPD — The time needed to read or write a memory double-word.

TOPi — The time needed to read or write a memory operand, where the operand size is given by the operation length of the instruction. It is always equivalent to either TOPB, TOPW or TOPD.

TCY — Internal processing overhead, in clock cycles.

L — Internal processing whose duration depends on the operation length. The number of clock cycles is derived by multiplying this value by the number of bytes in the operation length.

B.3.3 Equations

TOPB
If operand is in a register or is immediate then TOPB = 0
else TOPB = 3

TOPW
If operand is in a register or is immediate then TOPW = 0
else if word-aligned (even address) then TOPW = 3
else TOPW = 7

TOPD
If operand is in a register or is immediate then TOPD = 0
else if word-aligned (even address) then TOPD = 7
else TOPD = 11

TOPI
If operand is in a register or is immediate then TOPi = 0
else if i = byte then TOPi = TOPB
else if i = word then TOPi = TOPW
else (i = double-word) then TOPi = TOPD

TCY
TCY = 1

L
If i (operation length) = byte then L = 1
else if i = word then L = 2
else (i = double-word) L = 4

TEA
If REGISTER addressing then TEA = 2
If IMMEDIATE or ABSOLUTE addressing then TEA = 4
If REGISTER RELATIVE or MEMORY SPACE addressing then TEA = 5
If MEMORY RELATIVE addressing then TEA = 7 + TOPD
If TOP OF STACK addressing then
If access class = write then TEA = 4
If access class = read then TEA = 2
else TEA = 3
If EXTERNAL addressing then TEA = 11 + 2 * TOPD
If SCALED INDEXED addressing then TEA = TI1 + TI2
where TI1 depends on scale factor:
If byte indexing TI1 = 5
if word indexing TI1 = 7
if double-word indexing TI1 = 8
if quad-word indexing TI1 = 10
and TI2 = TEA of the basemode except:
if basemode is REGISTER then TI2 = 5
if basemode is TOP OF STACK then TI2 = 4

2.0 Architectural Description (Continued)

2.2.1 Addressing Mapping

Besides addressing the 16 Mbyte (24-bit address) external memory space, the NS32FX16 can also address 4 Gbytes (32-bit address) of its on-chip memory space *(see Figure 2.4)*. However, the upper 8 address bits are not issued to the memory or I/O outside the microprocessor. They are used only on-chip to access memory-mapped I/O (RAM and registers). This I/O is mapped in the FFFF0000-FFFFFFFF (hex) address range, which is part of the dedicated address space defined for National's Embedded System Processor architecture. The address space FFFFD000-FFFFDFFF (hex) is dedicated for the FAX Accelerator.

When accessing external memory, the address bits 24 to 31 (available on-chip only) should be kept zero.

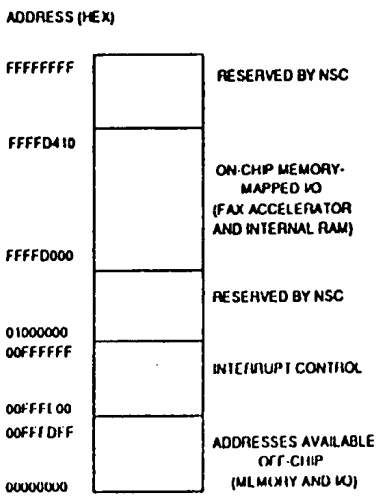


FIGURE 2-4. NS32FX16 Memory Organization

2.2.2 Dedicated Tables

Two of the NS32FX16 dedicated registers (MOD and INTBASE) serve as pointers to dedicated tables memory.

The INTBASE register points to the Interrupt Dispatch and Cascade Tables. These are described in Section 3.7.

The MOD register contains a pointer into the Module Table, whose entries are called Module Descriptors. A Module Descriptor contains four pointers, three of which are used by the NS32FX16. The MOD register contains the address of the Module Descriptor for the currently running module. It is automatically updated by the Call External Procedure instructions (CXP and CXPD).

The format of a Module Descriptor is shown in *Figure 2-5*. The Static Base entry contains the address of static data assigned to the running module. It is loaded into the CPU Static Base register by the CXP and CXPD instructions. The Program Base entry contains the address of the first byte of instruction code in the module. Since a module may have multiple entry points, the Program Base pointer serves only as a reference to find them.

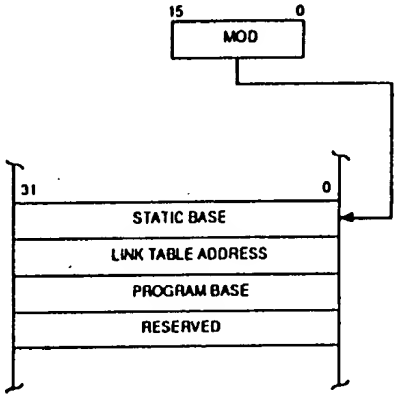


FIGURE 2-5. Module Descriptor Format

The Link Table Address points to the Link Table for the currently running module. The Link Table provides the information needed for:

1) Sharing variables between modules. Such variables are accessed through the Link Table via the External addressing mode.
2) Transferring control from one module to another. This is done via the Call External Procedure (CXP) instruction.

The format of a Link Table is given in *Figure 2-6*. A Link Table Entry for an external variable contains the 32-bit address of the variable. An entry for an external procedure contains two 16-bit fields: Module and Offset. The Module field contains the new MOD register contents for the module being entered. The Offset field is an unsigned number giving the position of the entry point relative to the new module's Program Base pointer.

For further details of the functions of these tables, see the *Series 32000 Instruction Set Reference Manual*.

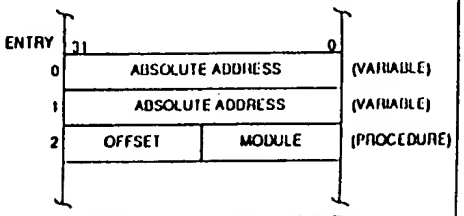


FIGURE 2-6. A Sample Link Table

2.3 INSTRUCTION SET

2.3.1 General Instruction Format

Figure 2-1 shows the general format of National's Embedded System Processor Instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing Mode ("Gen") fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

Index Bytes appear when either or both Gen fields specify Scaled Index. In this case, the Gen field specifies only the Scale Factor (1, 2, 4 or 8), and the Index Byte specifies which General Purpose Register to use as the Index, and which addressing mode calculation to perform before indexing. See *Figure 2-8*. Following the Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes.

Each Disp/Imm field may contain one of two displacements, or one immediate value. The size of a Displacement field is encoded within the top bits of that field, as shown in *Figure 2-9*, with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the Opcode field. Both Displacement and Immediate fields are stored most-significant byte first.

Note that this is different from the memory representation of data (Section 2.2).

Some instructions require additional "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition (Section 2.3.3).

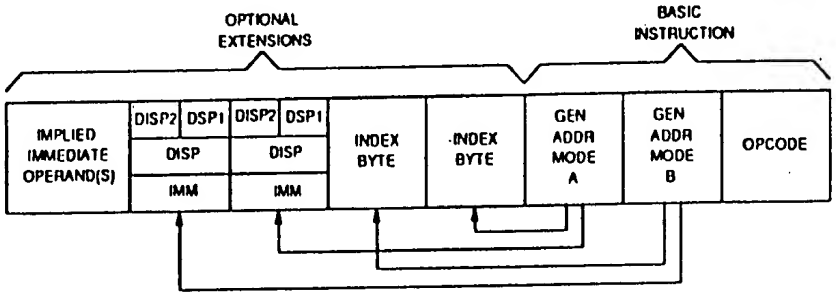


FIGURE 2-7. General Instruction Format

TABLE B-3. Average Execution with Wait States

INSTRUCTION	NUMBER OF CYCLES
BBOR	42 + ((107 + 2 * Twaitblt) + (44 + Twaitblt) * (*width* - 2)) * *height*
BBXOR	44 + ((107 + 2 * Twaitblt) + (44 + Twaitblt) * (*width* - 2)) * *height*
BBAND	45 + ((111 + 2 * Twaitblt) + (44 + Twaitblt) * (*width* - 2)) * *height*
BBFOR	48 + ((74 + 2 * Twaitblt) + (32 + Twaitblt) * (*width* - 2)) * *height*
BBSTOU	66 + ((170 + 2 * Twaitblt) + (60 + Twaitblt) * (*width* - 2)) * *height*
BITWT	shift = 0, 16 + Twaitrds + Twaitrdd + Twaitwrd shift = 1 - 8, 28 + Twaitblt
EXTBLT	35 + (19 + (12 + (Twaitrds + Twaitrdd + Twaitwrd))* *width*)* *height* (pre-read) 35 + (13 + (12 + (Twaitrds + Twaitrdd + Twaitwrd))* *width*)* *height* (no pre-read)

MOVMPB,W	if (Twaitwr > 1)
	16 + 7 * R2 + (Twaitwr - 1) * R2
	else
	16 + 7 * R2 + 0

MOVMPD	16 + 8 * R2 + Twaitwr * R2
TBIT	(27 + Twaitrd) per bit tested
SBITS	if R2 <= 25 39 + (2 * Twaitrdd + 2 * Twaitwrd + 2 * Twaitrds) else 42 + (2 * Twaitrdd + 2 * Twaitrds)
SBITP	8 + (34 * R2) + ((Twaitrdd + Twaitwrd) * R2)
MOVSB	if Twaitwr > 12 59+(14*R0)+(2*R0*4)+((Twaitwr-12)+Twaitrd)*R0 else 59+(14*R0)+(2*R0*4)+(Twaitrd*R0)
MOVSW	if Twaitwr > 12 59+(14*R0)+(2*R0*4)+((Twaitwr-12)+Twaitrd)*R0 else 59+(14*R0)+(2*R0*4)+(Twaitrd*R0)
MOVSD	if Twaitwr > 4 59+(10*R0)+(2*R0*8)+(((Twaitwr-4)*2)+(Twaitrd*2))*R0 else 59+(10*R0)+(2*R0*8)+((Twaitrd*2)*R0)

B.3 NS32FX16 GENERAL INSTRUCTION TIMING

B.3.1 Assumptions

The entire instruction, with all displacements and immediate operands, is assumed to be present in the instruction queue when needed.

Interference from instruction prefetches, which is very dependent upon the preceding instruction(s), is ignored. This assumption tends to affect the timing estimate in an optimistic direction.

It is assumed that all memory operand transfers are completed before the next instruction begins execution. In the case of an operand of access class rmw in memory, this is pessimistic, as the Write transfer occurs in parallel with the execution of the next instruction.

It is assumed that there is no overlap between the fetch of an operand and the following sequences of microcode. This is pessimistic, as the fetch of Operand A generally occurs in parallel with the effective address calculation of Operand B, and the fetch of Operand B occurs in parallel with the execution phase of the instruction.

Where possible, the values of operands are taken into consideration when they affect instruction timing, and a range of times is given. Where this is not done, the worst case is assumed.

B.2.3 Calculating the Effects of Shift Values

The formulae in the table give the average execution time for the BitBLT instructions (BBOR, BBXOR, BBAND, BBFOR, BITWT) with a *shift* of 0 to 8 bits. The BITWT and BBFOR instructions, however, optimize a *shift* of 0 bits by reading and writing only a word of the destination data (16 bits). For *shifts* greater than 0, they still must read and write a double-word of data (32 bits). Note that the EXTBLT instruction is not affected by the *shift* amount.

Shifts of greater than 8 bits add 1 clock per bit of *shift* > 8, per word of data read. For example, the BBOR with a *shift* of 15 bits yields the following formula:

42 + (107 + 44 * (*width* - 2)) * *height* + ((*shift* - 8) * *width* * *height*)

Inserting the previous example of the 10 by 50 BBOR:

45 + (107 + 44 * (10 - 2)) * 50 + ((15 - 8) * 10 * 50)
or
42 + (107 + 352) * 50 + (7 * 500) = 26,492 clocks or 1.77 msec @ 15 MHz

This represents the "worst case" time for this instruction, since a *shift* of greater than 15 bits can be handled by moving the source and destination pointers by 2 bytes and adjusting the *shift* amount.

The "best case" and "average case" times for most instructions are the same, due to reading the destination data during the *shifting* of the source data. This parallel operation is a feature of National's Embedded System Processor.

The "best case" for the BITWT and BBFOR instructions is a *shift* of zero bits. This is due to further internal optimization of these instructions, realizing that only a word of the destination data needs to be operated on if a *shift* of zero is specified.

Table B-2 shows the expected timing information for *shifts* greater than 8 bits.

TABLE B-2. Shift > 8

INSTRUCTION	NUMBER OF CYCLES
BBOR	42 + (107 + 44 * (*width* - 2)) * *height* + ((shift - 8) * *width* * *height*)
BBXOR	44 + (107 + 44 * (*width* - 2)) * *height* + ((shift - 8) * *width* * *height*)
BBAND	45 + (111 + 44 * (*width* - 2)) * *height* + ((shift - 8) * *width* * *height*)
BBFOR	48 + (74 + 32 * (*width* - 2)) * *height* + ((shift - 8) * *width* * *height*)
BBSTOD	66 + (170 + 60 * (*width* - 2)) * *height* + ((shift - 8) * *width* * *height*)
BITWT	28 + (*shift* - 8)
EXTBLT	35 + (17 + 13 * *width*) * *height* (pre-read) 35 + (11 + 13 * *width*) * *height* (no pre-read)

B.2.4 Calculating the Effects of Wait States

Since the new NS32FX16 instructions make use of the pipelined reads and writes of National's Embedded System Processor, calculation of the effect of wait states is rather difficult. As an example, in the MOVSi instruction group, each wait state on read operations adds 1 clock cycle per read bus access. Each wait state on write operations subtracts 1 clock cycle per write bus access from the TCY of the instruction, since updating the pointers occurs in parallel with the write operation. This means that wait states can be added to write cycles without changing the execution time of the instruction, up to a maximum of 13 wait states on writes for MOVSB and MOVSW, and 4 wait states on writes for MOVSD. At zero wait states, a MOVSD of 1,056,000 bytes (264,000 double-words) executes in:

30 + (10 * 264,000) + (2 * 264,000 * 8)
or
30 + 2,640,000 + 4,224,000 = 6,864,030 clocks or 458 msec @ 15 MHz

With two wait states on read and write (Twaitrd = 2 and Twaitwr = 2) — see Table B-3 — the time becomes:

30 + (10 * 264,000) + (2 * 264,000 * 8) + 0 + (2 * 2) * 264,000
or
30 + 2,640,000 + 4,224,000 + 1,056,000 = 7,920,030 clocks or 528 msec @ 15 MHz

Instructions that have *shift* amounts, such as BBOR, BBXOR, BBAND, BBFOR and BITWT, make use of the parallel nature of National's Embedded System Processor by doing the actual *shift* during the reading of the double-word destination data. This means that if there are wait states on read operations, these instructions are able to *shift* further, without impacting the overall time. For example, the total execution time for a BBFOR operation, *shifting* 8 bits with 2 wait states on read operations, is the same execution time as for a BBFOR operation *shifting* by 12 bits. This is because a destination read takes 4 clock cycles longer than a no-wait-state double-word read does. Note that this effect is not valid for more than 4 wait states, because at 4 wait states all possible *shift* values (0 - 15) are "hidden" during the destination read.

Note that in Table B-3, Twaitrds refers to the reading of the Source data, or of the table data used for a particular instruction. Twaitrdd refers to the reading of the Destination data, or of the data on which to be operated. Table B-3 shows the expected timing of instructions with wait states. Again, this is the average execution time, using a *shift* of eight (where appropriate).

Twaitblt is equal to (Twaitrds + 2 * Twaitrdd + 2 * Twaitwrd). This represents one BitBLT transfer, which makes the following equations easier.

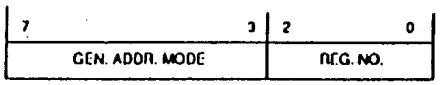


FIGURE 2-8. Index Byte Format

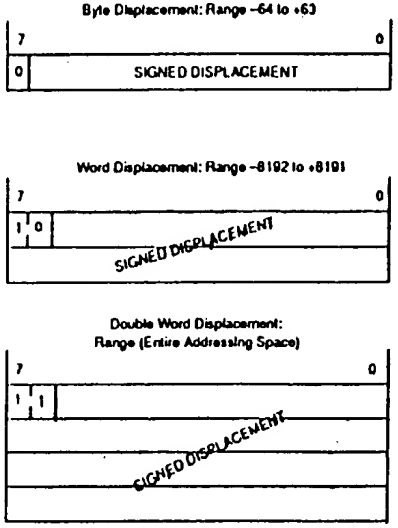


FIGURE 2-9. Displacement Encodings

2.3.2 Addressing Modes

The NS32FX16 CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode."

Addressing modes in the NS32FX16 are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode, within the instruction that acts upon that variable. Extraneous data movement is therefore minimized.

NS32FX16 Addressing Modes fall into nine basic types:

Register: The operand is available in one of the eight General Purpose Registers. In certain Slave Processor instructions, an auxiliary set of eight registers may be referenced instead.

Register Relative: A General Purpose Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.

Memory Space: Identical to Register Relative above, except that the register used is one of the dedicated registers PC, SP, SB or FP. These registers point to data areas generally needed by high-level languages.

Memory Relative: A pointer variable is found within the memory space pointed to by the SP, SB or FP register. A displacement is added to that pointer to generate the Effective Address of the operand.

Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.

Absolute: The address of the operand is specified by a displacement field in the instruction.

External: A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.

Top of Stack: The currently-selected Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.

Scaled Index: Although encoded as an addressing mode, Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any General Purpose Register by 1, 2, 4 or 8 and adding into the total, yielding the final Effective Address of the operand.

Table 2-1 is a brief summary of the addressing modes. For a complete description of their actions, see the *Series 32000 Instruction Set Reference Manual*.

In addition to the general modes, Register-Indirect with auto-increment/decrement and warps or pitch are available on several of the graphics instructions.

TABLE 2-1. NS32FX16 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0 or F0	None: Operand is in the specified register.
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R7 or F7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Relative			
10000	Frame memory relative	disp2(disp1 (FP))	Disp2 + Pointer; Pointer found at address Disp 1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1 (SP))	
10010	Static memory relative	disp2(disp1 (SB))	
Reserved			
10011	(Reserved for Future Use)		
Immediate			
10100	Immediate	value	None: Operand is input from instruction queue.
Absolute			
10101	Absolute	@disp	Disp.
External			
10110	External	EXT (disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top Of Stack			
10111	Top of stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register; "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	* + disp	
Scaled Index			
11100	Index, bytes	mode[Rn:B]	EA (mode) + Rn.
11101	Index, words	mode[Rn:W]	EA (mode) + 2 x Rn.
11110	Index, double words	mode[Rn:D]	EA (mode) + 4 x Rn.
11111	Index, quad words	mode[Rn:Q]	EA (mode) + 8 x Rn. "Mode" and "n" are contained within the Index Byte. EA (mode) denotes the effective address generated using mode.

B.1 INTRODUCTION

This chapter shows the expected timing of the NS32FX16 graphics instructions. Refer to Section B.3.

As this is advance information, it is subject to change without notice.

B.2 ASSUMPTIONS

The cycle time is one T-state of the Series 32000, equivalent to one-half of the input clock frequency present on the OSCIN pin of the NS32FX16. A 30-MHz clock source, therefore, yields a cycle time of 66.67 ns. Since the C and M bits of the NS32FX16's configuration register control an on-chip clock divider, setting those bits divides the clock by 1, 2, 4 or 8 to give a cycle time (with a 30-MHz clock source) of 66.67, 133.3 ns, 266.7 ns and 533.3 ns. This first section describes timing of the graphics instructions.

When needed, the entire instruction is assumed to be present in the instruction queue. Interference from instruction prefetches is ignored, with the exception of the BITWT instruction, where a no-wait-state prefetch is included (four clock cycles).

It is assumed that all memory operand transfers are completed before the next instruction begins execution. In the case of an operand of access class RMW in memory, this is pessimistic, as the write transfer occurs in parallel with the execution of the next instruction.

Where possible, the values of operands are taken into consideration when they affect instruction timing, and a range of times is given. Where this is not done, the average case is assumed. All memory accesses are assumed to be word aligned. Non-word-aligned data is acceptable but causes the execution time of a given instruction to increase.

The variety of definitions that follows allows accurate prediction of system performance when, for example, the source data may be in ROM and the destination may be in RAM, each having a different number of wait states. The number of wait states refers to the number of additional clock cycles requested via the $\overline{CWAIT}$ or $\overline{WAIT}$ pins of the NS32FX16, on a given byte or word-memory access.

B.2.1 Definitions

Twaitrd The number of wait states applied for a Read operation.

Twaitwr The number of wait states applied for a Write operation.

Twaitrds The number of wait states applied for a Read operation on source data. This also refers to the number of wait states applied for a table memory access (in the SBITS instruction, for example).

Twaitrdd The number of wait states applied for a Re operation on destination data.

Twaitwrd The number of wait states applied for a W operation on destination data.

Twaitblt Twaitrds + Twaitrdd * 2 + Twaitwrd ' the value used for BitBLT timing.

width The width of a BitBLT operation, in words.

height The height of a BitBLT operation, in s lines.

shift The number of bits of shift applied.

The Average Execution Time table (see Table B-1) li the execution time for the NS32FX16 instructions bas on an average shift of eight bits and a no-wait-st system design. The "no option" of each instruction used.

TABLE B-1. Average Execution Time

INSTRUCTION	NUMBER OF CYCLES
BBOR	42 + (107 + 44 * (*width* - 2)) * *height*
BBXOR	44 + (107 + 44 * (*width* - 2)) * *height*
BBAND	45 + (111 + 44 * (*width* - 2)) * *height*
BBFOR	48 + (61 + 25 * (*width* - 2)) * *height* fc *shift*=0
	48 + (74 + 32 * (*width* - 2)) * *height* fc *shift*=1-8
BBSTOD	66 + (170 + 60 * (*width* - 2)) * *height*
BITWT	if *shift* = 0, 16
	if *shift* = 1 - 8, 28
EXTBLT	35 + (19 + 12 * *width*) * *height* (pre-read)
	35 + (13 + 12 * *width*) * *height* (no pre-read)
MOVMPB,W	16 + 7 * R2
MOVMPD	16 + 8 * R2
TBIT	27 per bit tested
SBITS	if R2 < = 25
	39
	else
	42
SBITP	8 + (34 * R2)
MOVSB	59 + (14 * R0) + (2 * R0 * 4)
MOVSW	59 + (14 * R0) + (2 * R0 * 4)
MOVSD	59 + (10 * R0) + (2 * R0 * 8)

B.2.2 Interpreting the Table

To calculate the execution time for a given instructi complete the formula and evaluate. For example, calculate the time for a 10-word wide, 50-line high BB operation, the completed formula is:

42 + (107 + 44 * (10 - 2)) * 50

or

42 + (107 + 352) * 50 = 22,992 clocks or 1.53 msec at MHz, 0 wait states.

Appendix A: Instruction Formats (Continued)

Format 17

Trap (UND) Always

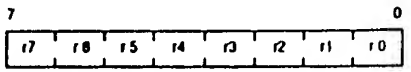
Format 18

Trap (UND) Always

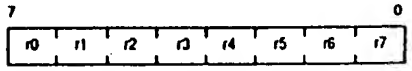
Format 19

Trap (UND) Always

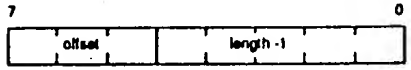
Implied Immediate Encodings:



Register Mask, appended to SAVE, ENTER



Register Mask, appended to RESTORE, EXIT



Offset/Length Modifier appended to INNS, EXTS

Note 1: Opcode not defined, CPU treats like MOVI. First operand has access class of read; second operand has access class of write, f-field selects 32-bit or 64-bit data.

Note 2: Opcode not defined; CPU treats like ADDI. First operand has access class of read; second operand has access class of read-modify-write; f-field selects 32-bit or 64-bit data.

Note 3: Opcode not defined; CPU treats like CMPI. First operand has access class of read, second operand has access class of read, f-field selects 32-bit or 64-bit data.

2.0 Architectural Description (Continued)

2.3.3 Instruction Set Summary

Table 2-2 presents a brief description of the NS32FX16 instruction set. The Format column refers to the Instruction Format tables (Appendix A). The Instruction column gives the instruction as coded in assembly language, and the Description column provides a short description of the function provided by that instruction. Further details of the exact operations performed by each instruction may be found in the *Series 32000 Instruction Set Reference Manual* and the NS32CG16 Printer/Display Processor Programmer's Reference.

Notations:

i = Integer length suffix: B = Byte, W = Word, D = Double Word

f = Floating Point length suffix: F = Standard Floating, L = Long Floating

gen = General operand. Any addressing mode can be specified.

short = A 4-bit value encoded within the Basic Instruction (see Appendix A for encodings).

Imm = Implied immediate operand. An 8-bit value appended after any addressing extensions.

disp = Displacement (addressing constant): 8, 16 or 32 bits. All three lengths legal.

reg = Any General-Purpose Register: R0-R7.

areg = Any Processor Register: SP, SB, FP, INTBASE, MOD, PSR, US (bottom 8 PSR bits).

cond = Any condition code, encoded as a 4-bit field within the Basic Instruction (see Appendix A for encodings).

TABLE 2-2. NS32FX16 Instruction Set Summary

MOVES

Format	Operation	Operands	Description
4	MOVi	gen,gen	Move a value.
2	MOVQi	short,gen	Extend and move a signed 4-bit constant.
7	MOVMi	gen,gen,disp	Move multiple: disp bytes (1 to 16).
7	MOVZBW	gen,gen	Move with zero extension.
7	MOVZiD	gen,gen	Move with zero extension.
7	MOVXBW	gen,gen	Move with sign extension.
7	MOVXiD	gen,gen	Move with sign extension.
4	ADDR	gen,gen	Move effective address.

INTEGER ARITHMETIC

Format	Operation	Operands	Description
4	ADDi	gen,gen	Add.
2	ADDQi	short,gen	Add signed 4-bit constant.
4	ADDCi	gen,gen	Add with carry.
4	SUBi	gen,gen	Subtract.
4	SUBCi	gen,gen	Subtract with carry (borrow).
6	NEGi	gen,gen	Negate (2's complement).
6	ABSi	gen,gen	Take absolute value.
7	MULi	gen,gen	Multiply.
7	QUOi	gen,gen	Divide, rounding toward zero.
7	REMi	gen,gen	Remainder from QUO.
7	DIVi	gen,gen	Divide, rounding down.
7	MODi	gen,gen	Remainder from DIV (Modulus).
7	MEIi	gen,gen	Multiply to extended integer.
7	DEIi	gen,gen	Divide extended integer.

PACKED DECIMAL (BCD) ARITHMETIC

Format	Operation	Operands	Description
6	ADDPi	gen,gen	Add packed.
6	SUBPi	gen,gen	Subtract packed.

TABLE 2-2. NS32FX16 Instruction Set Summary (Continued)

INTEGER COMPARISON

Format	Operation	Operands	Description
4	CMPi	gen,gen	Compare.
2	CMPQi	short,gen	Compare to signed 4-bit constant.
7	CMPM	gen,gen,disp	Compare multiple: disp bytes (1 to 16).

LOGICAL AND BOOLEAN

Format	Operation	Operands	Description
4	ANDi	gen,gen	Logical AND.
4	ORi	gen,gen	Logical OR.
4	BICi	gen,gen	Clear selected bits.
4	XORi	gen,gen	Logical exclusive OR.
6	COMi	gen,gen	Complement all bits.
6	NOTi	gen,gen	Boolean complement: LSB only.
2	Scondi	gen	Save condition code (cond) as a Boolean variable of size i.

SHIFTS

Format	Operation	Operands	Description
6	LSHi	gen,gen	Logical shift, left or right.
6	ASHi	gen,gen	Arithmetic shift, left or right.
6	ROTi	gen,gen	Rotate, left or right.

BIT FIELDS

Bit fields are values in memory that are not aligned to byte boundaries. Examples are PACKED arrays and records used in Pascal. "Extract" instructions read and align a bit field. "Insert" instructions write a bit field from an aligned source.

Format	Operation	Operands	Description
8	EXTi	reg,gen,gen,disp	Extract bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented)
7	EXTSi	gen,gen,imm,imm	Extract bit field (short form).
7	INSSi	gen,gen,imm,imm	Insert bit field (short form).
8	CVTP	reg,gen,gen	Convert to bit field pointer.

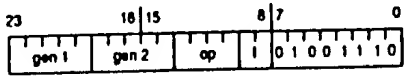
ARRAYS

Format	Operation	Operands	Description
8	CHECKi	reg,gen,gen	Index bounds check.
8	INDEXi	reg,gen,gen	Recursive indexing step for multiple-dimensional arrays.

Format 5

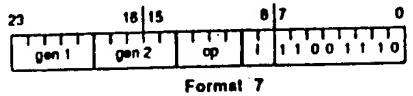
MOVS	-0000	BITWT	-1000
CMPS	-0001	TBITS	-1001
SETCFG	-0010	BBAND	-1010
SKPS	-0011	SBITPS	-1011
BBSTOD	-0100	BBFOR	-1100
EXTBLT	-0101	SBITS	-1101
BBOR	-0110	BBXOR	-1110
MOVMP	-0111		

No Operation on 1111



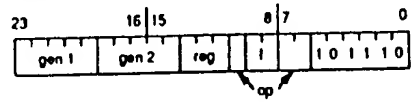
Format 6

ROT	-0000	NEG	-1000
ASH	-0001	NOT	-1001
CBIT	-0010	Trap(UND)	-1010
CBITI	-0011	SUBP	-1011
Trap (UND)	-0100	ABS	-1100
LSH	-0101	COM	-1101
SBIT	-0110	IBIT	-1110
SBITI	-0111	ADDP	-1111



Format 7

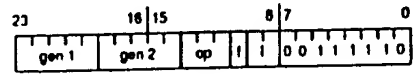
MOVM	-0000	MUL	-1000
CMPM	-0001	MEI	-1001
INSS	-0010	Trap(UND)	-1010
EXTS	-0011	DEI	-1011
MOVXBW	-0100	QUO	-1100
MOVZBW	-0101	REM	-1101
MOVZiD	-0110	MOD	-1110
MOVXiD	-0111	DIV	-1111



Format 8

EXT	-0 00	INDEX	-1 00
CVTP	-0 01	FFS	-1 01
INS	-0 10		
CHECK	-0 11		

Trap (UND) on -1 10 and -1 11

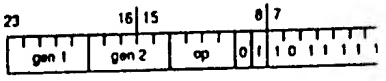


Format 9

MOVif	-000	ROUND	-100
LFSR	-001	TRUNC	-101
MOVLF	-010	SFSR	-110
MOVFL	-011	FLOOR	-111

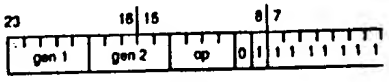
Format 10

Trap (UND) Always



Format 11

ADDf	-0000	DIVf	-10(
MOVf	-0001	(Note 1)	-10(
CMPf	-0010	Trap (UND)	-101
(Note 3)	-0011	Trap (UND)	-101
SUBf	-0100	MULf	-11(
NEGf	-0101	ABSf	-11(
Trap (UND)	-0110	Trap (UND)	-111
Trap (UND)	-0111	Trap (UND)	-111



Format 12

(Note 2)	-0000	(Note 2)	-100(
(Note 1)	-0001	(Note 1)	-100
POLFf	-0010	Trap (UND)	-101(
DOTf	-0011	Trap (UND)	-101
SCALBf	-0100	(Note 2)	-110(
LOGBf	-0101	(Note 1)	-110
Trap (UND)	-0110	Trap (UND)	-111(
Trap (UND)	-0111	Trap (UND)	-111

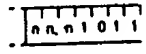
* Instructions with Format 12 are available only when the NS3238 is used.

Format 13

Trap (UND) Always

Format 14

Trap (UND) Always



Format 15

Trap (UND) Always

Format 16

Trap (UND) Always

Appendix A: Instruction Formats

NOTATIONS

i = Integer Type Field
B = 00 (Byte)
W = 01 (Word)
D = 11 (Double Word)

f = Floating-Point Type Field
F = 1 (Std. Floating: 32 bits)
L = 0 (Long Floating: 64 bits)

op = Operation Code
Valid encoding shown with each format.

gen 1, gen 2 = General Addressing Mode Field.
See Section 2.3.2 for encodings.

reg = General Purpose Register Number

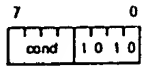
cond = Condition Code Field
0000 = EQual: Z = 1
0001 = Not Equal: Z = 0
0010 = Carry Set: C = 1
0011 = Carry Clear: C = 0
0100 = Higher: L = 1
0101 = Lower or Same: L = 0
0110 = Greater Than: N = 1
0111 = Less or Equal: N = 0
1000 = Flag Set: F = 1
1001 = Flag Clear: F = 0
1010 = LOwer: L = 0 and Z = 0
1011 = Higher or Same: L = 1 or Z = 1
1100 = Less Than: N = 0 and Z = 0
1101 = Greater or Equal: N = 1 or Z = 1
1110 = (Unconditionally True)
1111 = (Unconditionally False)

short = Short Immediate value. May contain
quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
cond: Condition Code (above), in Scond.
areg: CPU Dedicated Register, in LPR, SPR.
0000 = US
0001 - 0111 = (Reserved)
1000 = FP
1001 = SP
1010 = SB
1011 = (Reserved)
1100 = (Reserved)
1101 = PSR
1110 = INTBASE
1111 = MOD

Options: in String Instructions

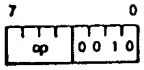
T = Translated
B = Backward
U/W = 00: None
01: While Match
11: Until Match

Configuration bits in SETCFG instruction



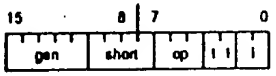
Format 0

Bcond	(BR)		



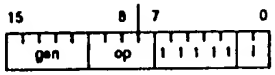
Format 1

BSR	-0000	ENTER	-1000
RET	-0001	EXIT	-1001
CXP	-0010	NOP	-1010
RXP	-0011	WAIT	-1011
RETT	-0100	DIA	-1100
RETI	-0101	FLAG	-1101
SAVE	-0110	SVC	-1110
RESTORE	-0111	BPT	-1111



Format 2

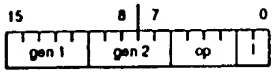
ADDQ	-000	ACB	-100
CMPQ	-001	MOVQ	-101
SPR	-010	LPR	-110
Scond	-011		



Format 3

CXPD	-0000	ADJSP	-1010
BICPSR	-0010	JSR	-1100
JUMP	-0100	CASE	-1110
BISPSR	-0110		

Trap (UND) on XXX1, 1000



Format 4

ADD	-0000	SUB	-1000
CMP	-0001	ADDR	-1001
BIC	-0010	AND	-1010
ADDC	-0100	SUBC	-1100
MOV	-0101	TBIT	-1101
OR	-0110	XOR	-1110

TABLE 2-2. NS32FX16 Instruction Set Summary (Continued)

STRINGS

String instructions assign specific functions to the General Purpose Registers:
R4 — Comparison Value
R3 — Translation Table Pointer
R2 — String 2 Pointer
R1 — String 1 Pointer
R0 — Limit Count

Options on all string instructions are:
B (Backward): Decrement string pointers after each step rather than incrementing.
U (Until match): End instruction if String 1 entry matches R4.
W (While match): End instruction if String 1 entry does not match R4.
All string instructions end when R0 decrements to zero.

Format	Operation	Operands	Description
5	MOVSi	options	Move string 1 to string 2.
	MOVST	options	Move string, translating bytes.
5	CMPSi	options	Compare string 1 to string 2.
	CMPST	options	Compare, translating string 1 bytes.
5	SKPSi	options	Skip over string 1 entries.
	SKPST	options	Skip, translating bytes for until/while.

JUMPS AND LINKAGE

Format	Operation	Operands	Description
3	JUMP	gen	Jump.
0	BR	disp	Branch (PC Relative).
0	Bcond	disp	Conditional branch.
3	CASEi	gen	Multiway branch.
2	ACBi	short,gen,disp	Add 4-bit constant and branch if non-zero.
3	JSR	gen	Jump to subroutine.
1	BSR	disp	Branch to subroutine.
1	CXP	disp	Call external procedure.
3	CXPD	gen	Call external procedure using descriptor.
1	SVC		Supervisor call.
1	FLAG		Flag trap.
1	BPT		Breakpoint trap.
1	ENTER	[reg list],disp	Save registers and allocate stack frame (Enter Procedure).
1	EXIT	[reg list]	Restore registers and reclaim stack frame (Exit Procedure).
1	RET	disp	Return from subroutine.
1	RXP	disp	Return from external procedure call.
1	RETT	disp	Return from trap. (Privileged)
1	RETI		Return from interrupt. (Privileged)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save general purpose registers.
1	RESTORE	[reg list]	Restore general purpose registers.
2	LPRi	areg,gen	Load dedicated register. (Privileged if PSR or INTBASE)
2	SPRi	areg,gen	Store dedicated register. (Privileged if PSR or INTBASE)
3	ADJSPi	gen	Adjust stack pointer.
3	BISPSRi	gen	Set selected bits in PSR. (Privileged if not Byte length)
3	BICPSRi	gen	Clear selected bits in PSR. (Privileged if not Byte length)
5	SETCFG	[option list]	Set configuration register. (Privileged)

Table 2-2. NS32FX16 Instruction Set Summary (Continued)

FLOATING-POINT

Format	Operation	Operands	Description
11	MOVf	gen,gen	Move a floating-point value.
9	MOVLF	gen,gen	Move and shorten a long value to standard.
9	MOVFL	gen,gen	Move and lengthen a standard value to long.
9	MOVif	gen,gen	Convert any integer to standard or long floating.
9	ROUNDfi	gen,gen	Convert to integer by rounding.
9	TRUNCfi	gen,gen	Convert to integer by truncating, toward zero.
9	FLOORfi	gen,gen	Convert to largest integer less than or equal to value.
11	ADDf	gen,gen	Add.
11	SUBf	gen,gen	Subtract.
11	MULf	gen,gen	Multiply.
11	DIVf	gen,gen	Divide.
11	CMPf	gen,gen	Compare.
11	NEGf	gen,gen	Negate.
11	ABSf	gen,gen	Take absolute value.
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.
12	POLYf	gen,gen	Polynomial Step.
12	DOTf	gen,gen	Dot Product.
12	SCALBf	gen,gen	Binary Scale.
12	LOGBf	gen,gen	Binary Log.

MISCELLANEOUS

1	NOP		No operation.
1	WAIT		Wait for interrupt.
1	DIA		Diagnose. Single-byte "Branch to Self" for hardware breakpointing. Not for use in programming.

GRAPHICS

Format	Operation	Operands	Description
5	BBOR	options*	Bit-aligned block transfer 'OR'.
5	BBAND	options	Bit-aligned block transfer 'AND'.
5	BBFOR		Bit-aligned block transfer fast 'OR'.
5	BBXOR	options	Bit-aligned block transfer 'XOR'.
5	BBSTOD	options	Bit-aligned block source to destination.
5	BITWT		Bit-aligned word transfer.
5	EXTBLT	options	External bit-aligned block transfer.
5	MOVMPI		Move multiple pattern.
5	TBITS	options	Test bit string.
5	SBITS		Set bit string.
5	SBITPS		Set bit perpendicular string.

BITS

Format	Operation	Operands	Description
4	TBITi	gen,gen	Test bit.
6	SBITI	gen,gen	Test and set bit.
6	SBITIi	gen,gen	Test and set bit, interlocked.
6	CBITI	gen,gen	Test and clear bit.
6	CBITIi	gen,gen	Test and clear bit, interlocked.
6	IBITi	gen,gen	Test and invert bit.
8	FFSi	gen,gen	Find first set bit.

*Note: Options are controlled by fields of the instruction, PSR status bits, or dedicated register values.

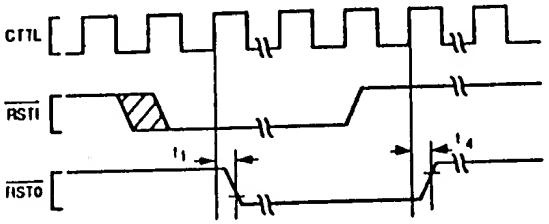


FIGURE 4-17. Non-Power-On Reset

Note 1: During Reset the HOLD signal must be kept high.
Note 2: After RSTI is deasserted the first bus cycle will be an instruction fetch at address zero.

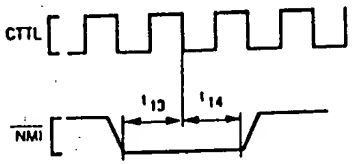


FIGURE 4-18. NMI Interrupt Signal Timing

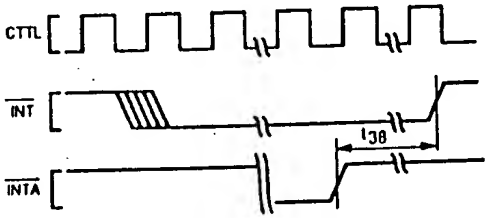


FIGURE 4-19. INT Interrupt Signal Detection

Note 1: Once INT is asserted, it must remain asserted until it is acknowledged.
Note 2: INTA is the Interrupt Acknowledge bus cycle (not a CPU signal). Refer to Section 3.4.1 and Table 3.4.

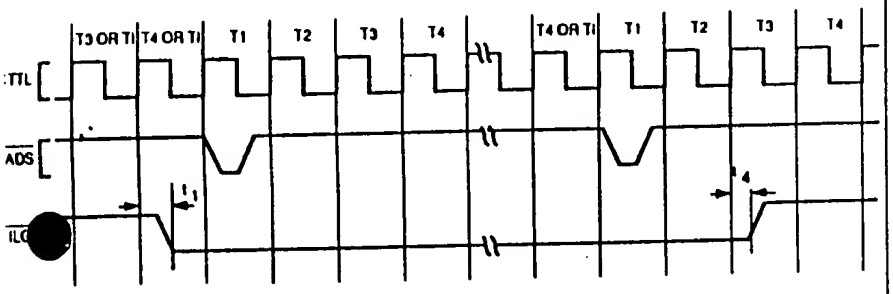


FIGURE 4-14. Interlocked Bus Cycle

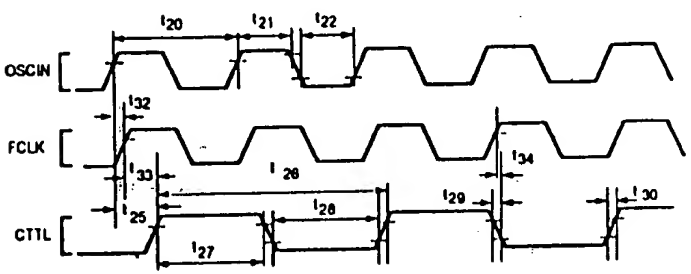


FIGURE 4-15. Clock Waveforms

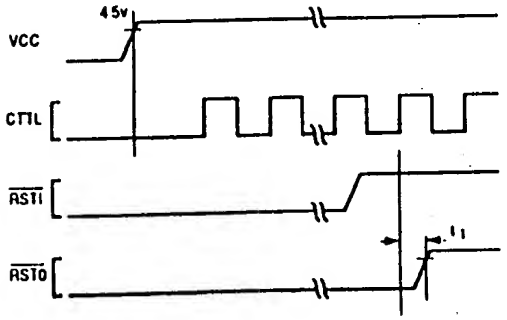


FIGURE 4-16. Power-On Reset

2.4 GRAPHICS SUPPORT

The following sections provide a brief description of the NS32FX16 graphics support capabilities. Basic discussions on frame buffer addressing and BitBLT operations are also provided. More detailed information on the NS32FX16 graphics support instructions can be found in the NS32CG16 Printer/Display Processor Programmer's Reference.

2.4.1 Frame Buffer Addressing

There are two basic addressing schemes for referencing pixels within the frame buffer: Linear and Cartesian (or x-y). Linear addressing associates a single number to each pixel representing the physical address of the corresponding bit in memory. Cartesian addressing associates two numbers to each pixel representing the x and y coordinates of the pixel relative to a point in the Cartesian space taken as the origin. The Cartesian space is generally defined as having the origin in the upper left. A movement to the right increases the x coordinate; a movement downward increases the y coordinate.

The correspondence between the location of a pixel in the Cartesian space and the physical (BIT) address in memory is shown in *Figure 2-10*. The origin of the Cartesian space (x = 0, y = 0) corresponds to the bit address 'ORG'. Incrementing the x coordinate increments the bit address by one. Incrementing the y coordinate increments the bit address by an amount representing the warp (or pitch) of the Cartesian space. Thus, the linear address of a pixel at location (x, y) in the Cartesian space can be found by the following expression.

ADDR = ORG + y * WARP + x

Warp is the distance (in bits) in the physical memory space between two vertically adjacent bits in the Cartesian space.

Example 1 below shows two NS32FX16 instruction sequences to set a single pixel given the x and y coordinates. Example 2 shows how to create a fat pixel by setting four adjacent bits in the Cartesian space.

Example 1: Set pixel at location (x, y)
Setup: R0 x coordinate
R1 y coordinate

Instruction Sequence 1:

```
MULD    WARP, R1        ; Y*WARP
ADDD    R0, R1          ; X+BIT OFFSET
SBITD   R1, ORG         ; SET PIXEL
```

Instruction Sequence 2:

```
INDEXD  R1, (WARP-1), R0  ; Y*WARP + X
SBITD   R1, ORG           ; SET PIXEL
```

Example 2: Create fat pixel by setting bits at locations (x, y), (x + 1, y), (x, y + 1) and (x + 1, y + 1).
Setup: R0 x coordinate
R1 y coordinate

Instruction Sequence:

```
INDEXD  R1, (WARP-1), R0  ; BIT ADDRESS
SBITD   R1, ORG           ; SET FIRST PIXEL
ADDQD   1, R1             ; (X+1, Y)
SBITD   R1, ORG           ; SECOND PIXEL
ADDD    (WARP-1), R1      ; (X, Y+1)
SBITD   R1, ORG           ; THIRD PIXEL
ADDQD   1, R1             ; (X+1, Y+1)
SBITD   R1, ORG           ; LAST PIXEL
```

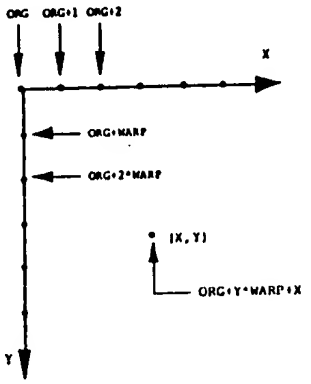


FIGURE 2-10. Correspondence between Linear and Cartesian Addressing

2.4.2 BitBLT Fundamentals

BitBLT, Bit-aligned BLock Transfer, is a general operator that provides a mechanism to move an arbitrary size rectangle of an image from one part of the frame buffer to another. During the data transfer process a bitwise logical operation can be performed between the source and the destination data. BitBLT is also called Raster-Op: operations on rasters. It defines two rectangular areas, source and destination, and performs a logical operation (e.g., AND, OR, XOR) between these two areas and stores the result back to the destination. It can be expressed in simple notation as:

Source op Destination -> Destination
op: AND, OR, XOR, etc.

2.4.2.1 Frame Buffer Architecture

There are two basic types of frame buffer architectures: plane-oriented or pixel-oriented. BitBLT takes advantage of the plane-oriented frame buffer architecture's attribute of multiple, adjacent pixels-per-word, facilitating the movement of large blocks of data. The source and destination starting addresses are expressed as pixel addresses. The width and height of the block to be moved are expressed in terms of pixels and scan lines. The source block may start and end at any bit position of any word, and the same applies for the destination block.

2.4.2.2 Bit Alignment

Before a logical operation can be performed between the source and the destination data, the source data must first be bit aligned to the destination data. In *Figure 2-11* the source data needs to be shifted three bits to the right in order to align the first pixel (i.e., the pixel at the top left corner) in the source data block to the first pixel in the destination data block.

2.4.2.3 Block Boundaries and Destination Masks

Each BitBLT destination scan line may start and end at any bit position in any data word. The neighboring bits (bits sharing the same word address with any words in the destination data block, but not a part of the BitBLT rectangle) of the BitBLT destination scan line must remain unchanged after the BitBLT operation.

Due to the plane-oriented frame buffer architecture, all memory operations must be word-aligned. In order to preserve the neighboring bits surrounding the BitBLT destination block, both a left mask and a right mask are needed for all the leftmost and all the rightmost data words of the destination block. The left mask and the right mask both remain the same during a BitBLT operation.

The following example illustrates the bit alignment requirements. In this example, the memory data path is 16 bits wide. *Figure 2.11* shows a 32 pixel by 32 scan line frame buffer which is organized as a long bit stream which wraps around every two words (32 bits). The origin (top left corner) of the frame buffer starts from the lowest word in memory (word address 00 (hex)).

Each word in the memory contains 16 bits, D0-D15. The least significant bit of a memory word, D0, is defined as the first displayed pixel in a word. In this example, BitBLT addresses are expressed as pixel addresses relative to the origin of the frame buffer. The source block starting address is 021 (hex) (the second pixel in the third word). The destination block starting address is 204 (hex) (the fifth pixel in the 33rd word). The block width is 14 (hex), and the height is 06 (hex) (corresponding to 6 scan lines). The shift value is 3.

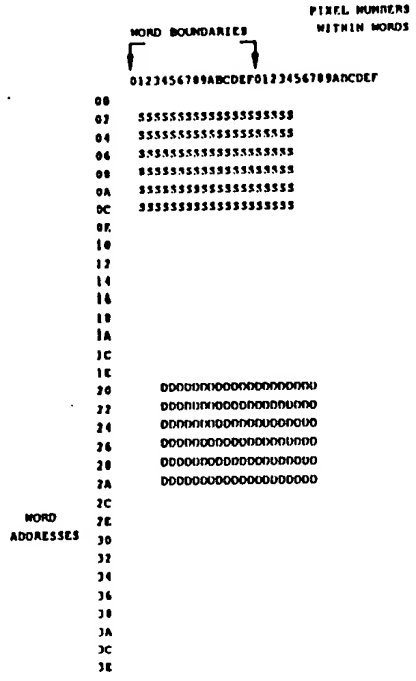


FIGURE 2-11. 32-Pixel by 32-Scan Line Frame Buffer

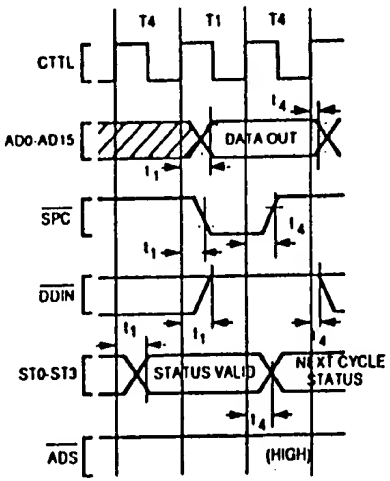


FIGURE 4-10. Slave Processor Write Timing

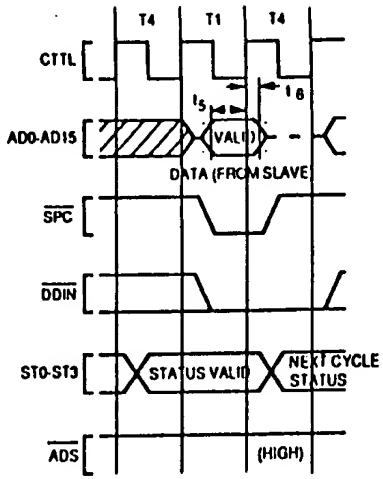


FIGURE 4-11. Slave Processor Read Timing

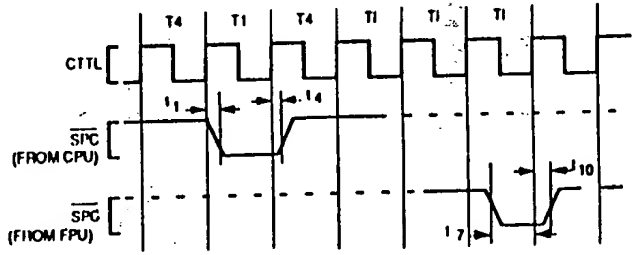


FIGURE 4-12. $\overline{SPC}$ Timing

There is a minimum one clock cycle between the $\overline{SPC}$ output asserted by the CPU and the $\overline{SPC}$ input from the FPU.

After transferring the last operand to the FPU, the CPU turns OFF the output driver and holds $\overline{SPC}$ high with an internal 5kΩ pullup.

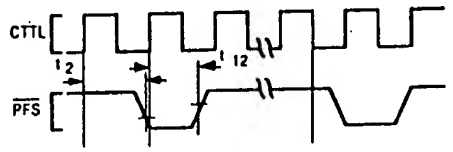


FIGURE 4-13. Relationship of $\overline{PFS}$ to Clock Cycles

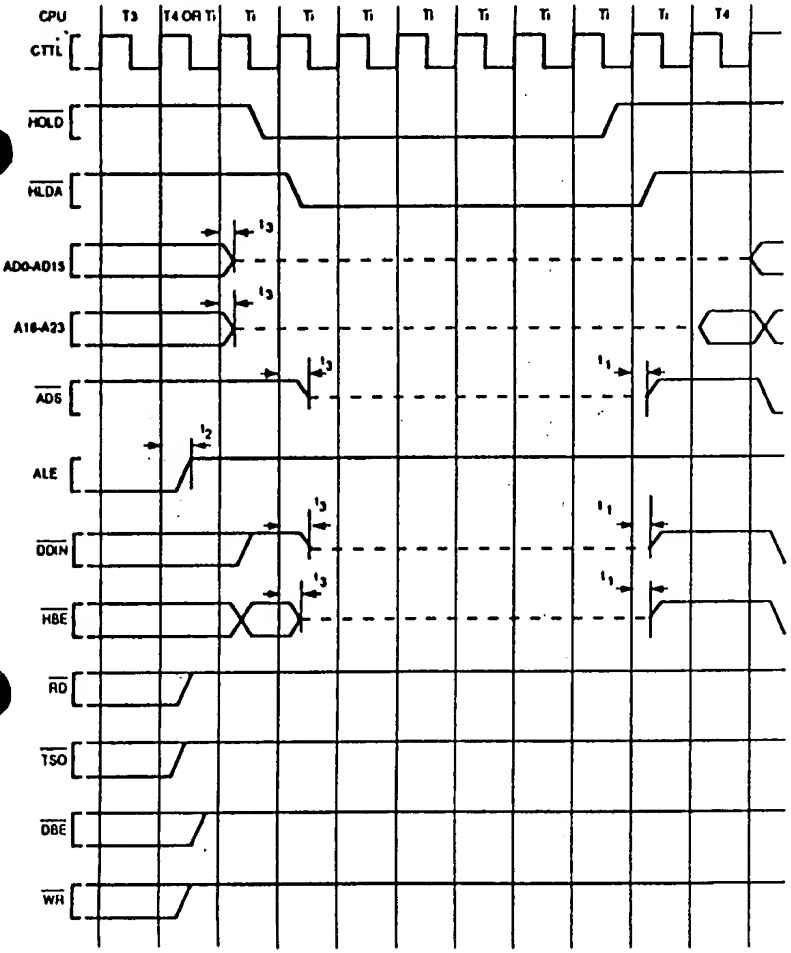


FIGURE 4-9. $\overline{\text{HOLD}}$ Acknowledge (Bus Initially Idle)

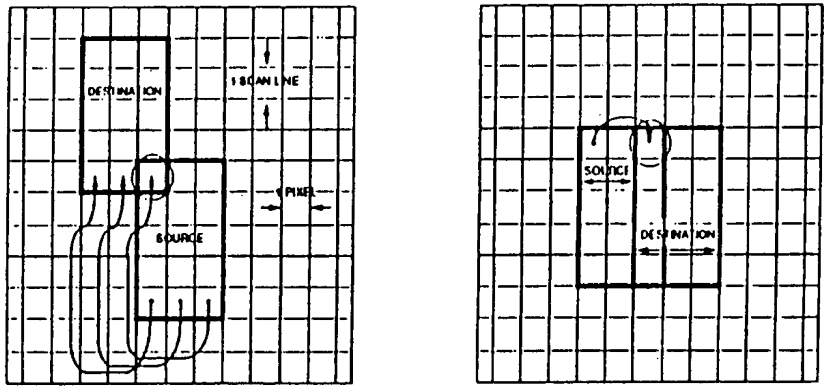


FIGURE 2-12. Overlapping BitBLT Blocks

The left mask and the right mask are 0000,1111,1111,1111 and 1111,1111,0000,0000 respectively.

Note 1: Zeroes in either the left mask or the right mask indicate the destination bits which will not be modified.

Note 2: The BB (function) and EXTBLT instructions use different setup parameters and techniques.

2.4.2.4 BitBLT Directions

A BitBLT operation moves a rectangular block of data in a frame buffer. The operation itself can be considered as a subroutine with two nested loops. The loops are preceded by setup operations. In the outer loop the source and destination starting addresses are calculated, and the test for completion is performed. In the inner loop the actual data movement for a single scan line takes place. The length of the inner loop is the number of (aligned) words spanned by each scan line. The length of the outer loop is equal to the height (number of scan lines) of the block to be moved. A skeleton of the subroutine representing the BitBLT operation follows:

BitBLT:	calculate BitBLT setup parameters; (once per BitBLT operation); such as width, height, bit misalignment (shift number), left, right masks, horizontal, vertical directions, etc.
OUTERLOOP:	calculate source, dest addresses; (once per scan line).
INNERLOOP:	move data (logical operation) and increment addresses; (once per word).
UNTIL	done horizontally.
UNTIL	done vertically.

Note: In the NS32FX16 only the setup operations must be done by the programmer. The inner and outer loops are automatically executed by the BitBLT instructions.

Each loop can be executed in one of two directions: the inner loop from left to right or right to left, the outer loop from top to bottom (down) or bottom to top (up).

The ability to move data starting from any corner of the BitBLT rectangle is necessary to avoid destroying the BitBLT source data as a result of destination writes when the source and destination are overlapped (i.e., when they share pixels). This situation is routinely encountered while panning or scrolling.

A determination of the correct execution directions of the BitBLT must be performed whenever the source and destination rectangles overlap. Any overlap will result in the destruction of source data (from a destination write) if the correct vertical direction is not used. Horizontal BitBLT direction is of concern only in certain cases of overlap, as will be explained below.

Figures 2-12(a) and *(b)* illustrate two cases of overlap. Here, the BitBLT rectangles are three pixels wide by five scan lines high; they overlap by a single pixel in *(a)* and a single column of pixels in *(b)*. For purposes of illustration, the BitBLT is assumed to be carried out pixel-by-pixel. This convention does not affect the conclusions.

In *Figure 2-12(a)*, if the BitBLT is performed in the UP direction (bottom-to-top), one of the transfers of the bottom scan line of the source will write to the circled pixel of the destination. Due to the overlap, this pixel is also part of the uppermost scan line of the source rectangle. Thus, data needed later is destroyed. Therefore, this BitBLT must be performed in the DOWN direction. Another example of this occurs any time the screen is moved in a purely vertical direction, as in scrolling text. It should be noted that, in both of these cases, the choice of horizontal BitBLT direction may be made arbitrarily.

Figure 2-12(b) demonstrates a case in which the horizontal BitBLT direction may not be chosen arbitrarily. This is an instance of purely horizontal movement of data (panning). Because the movement from source to destination involves data within the same scan line, the incorrect direction of movement will overwrite data which will be needed later. In this example, the correct direction is from right to left.

2.4.2.5 BitBLT Variations

The 'classical' definition of BitBLT, as described in "Smalltalk-80 The Language and its Implementation", by Adele Goldberg and David Robson, provides for three operands: source, destination and mask/texture. This third operand is commonly used in monochrome systems to incorporate a stipple pattern into an area. These stipple patterns provide the appearance of multiple shades of gray in single-bit-per-pixel systems, in a manner similar to the 'halftone' process used in printing.

Texture op1 Source op2 Destination → Destination

While the NS32CG160 and the external BPU (if used) are essentially two-operand devices, three-operand BitBLT operations can be implemented quite flexibly and efficiently by performing the two operations serially.

2.4.3 Graphics Support Instructions

The NS32FX16 provides eleven instructions for supporting graphics oriented applications. These instructions are divided into three groups according to the operations they perform. General descriptions for each of them and the related formats are provided in the following sections.

2.4.3.1 BitBLT (Bit-aligned BLock Transfer)

This group includes seven instructions. They are used to move characters and objects into the frame buffer which will be printed or displayed. One of the instructions works in conjunction with an external BitBLT Processing Unit (BPU) to maximize performance. The other six are executed by the NS32FX16.

Bit-aligned Block Transfer

Syntax: BB(function) Options

Setup:	R0	base address, source data
	R1	base address, destination data
	R2	shift value
	R3	height (in lines)
	R4	first mask
	R5	second mask
	R6	source warp (adjusted)
	R7	destination warp (adjusted)
	0(SP)	width (in words)
Function:	AND, OR, XOR, FOR, STOD	
Options:	IA	Increasing Address (default option). When IA is selected, scan lines are transferred in the increasing BIT/BYTE order.
	DA	Decreasing Address.
	S	True Source (default option).
	-S	Inverted Source.

These five instructions perform standard BitBLT operations between source and destination blocks. The operations available include the following:

BBAND:	src	AND	dst
	-src	AND	dst
BBOR:	src	OR	dst
	-src	OR	dst
BBXOR:	src	XOR	dst
	-src	XOR	dst
BBFOR:	src	OR	dst
BBSTOD:	src	TO	dst
	-src	TO	dst

'src' and '-src' stand for 'True Source' and 'Inverted Source' respectively; 'dst' stands for 'Destination'.

Note 1: For speed reasons, the BB instructions require the masks to be specified with respect to the source block. In *Figure 2-11* masking was defined relative to the destination block.

Note 2: The options -S and DA are not available for the BBFOR instruction.

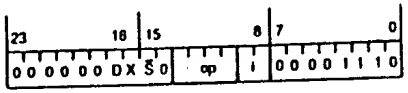
Note 3: BBFOR performs the same operation as BBOR with IA and S options.

Note 4: IA and DA are mutually exclusive and so are S and -S.

Note 5: The width is defined as the number of words of source data to read.

Note 6: An odd number of bytes can be specified for the source warp. However, word alignment of source scan lines will result in faster execution.

The horizontal and vertical directions of the BitBLT operations performed by the above instructions, with the exception of BBFOR, are both programmable. The horizontal direction is controlled by the IA and DA options. The vertical direction is controlled by the sign of the source and destination warps. *Figure 2-13* and Table 2-3 show the format of the BB instructions and the encodings for the 'op' and 'i' field



- D is set when the DA option is selected
- $\bar{S}$ is set when the -S option is selected
- X is set for BBAND, and it is clear for all other BB instructions

FIGURE 2-13. BB Instructions Format

TABLE 2-3. 'op' and 'i' Field Encodings

Instruction	Options	'op' Field	'i' Field
BBAND	Yes	1010	11
BBOR	Yes	0110	01
BBXOR	Yes	1110	01
BBFOR	No	1100	01
BBSTOD	Yes	0100	01

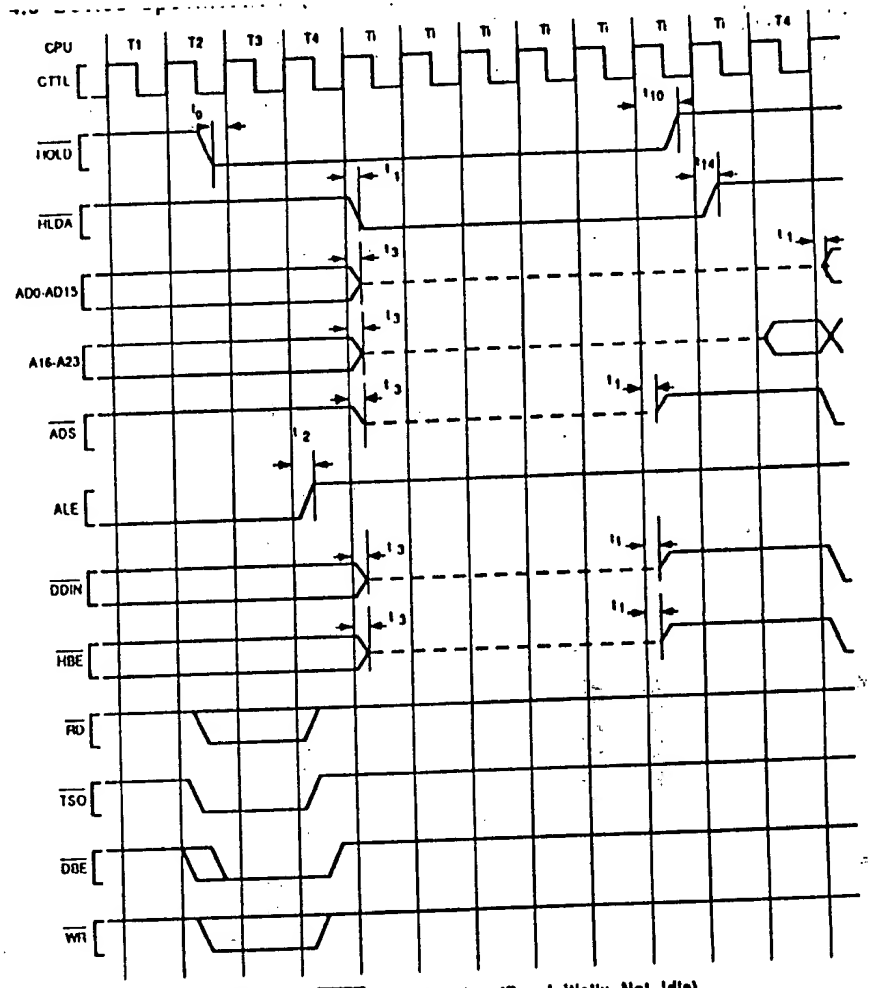


FIGURE 4-8. HOLD Acknowledge (Bus Initially Not Idle)

Note: When the bus is not idle, HOLD must be asserted before the rising edge of CTTL of the timing state that proceeds state T4 in order for the request to be acknowledged.

4.0 Device Specifications (Continued)

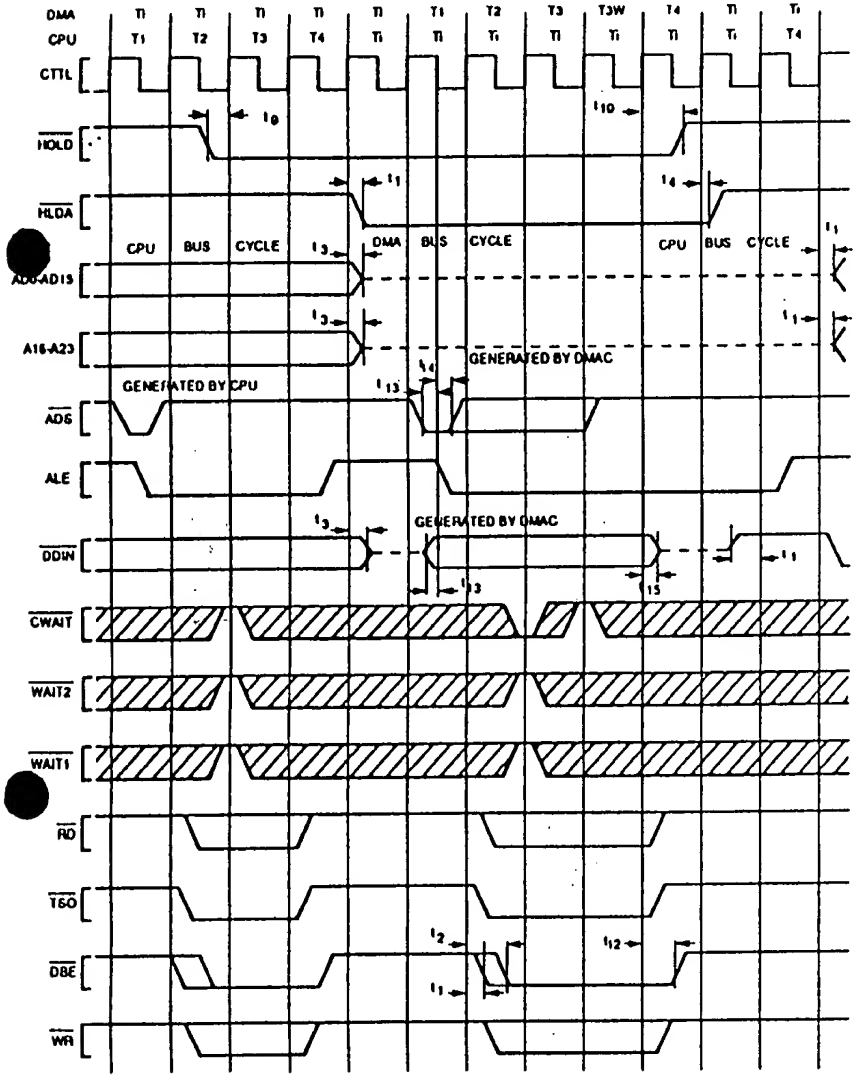


FIGURE 4-7. DMAC Initiated Bus Cycle

Note 1: $\overline{ADS}$ must be deactivated before state T4 of the DMA controller cycle

Note 2 During a DMA cycle $\overline{WAIT}$1-2 must be kept inactive unless they are monitored by the DMA controller. A DMA cycle is similar to a CPU cycle. The NS32FX16 generates $\overline{TSO}$, $\overline{RD}$, $\overline{WR}$ and $\overline{DBE}$. The DMAC drives the address/data lines HBE, ADS and DDIN

Note 3: During a DMA cycle, if the $\overline{ADS}$ signal is pulsed in order to initiate a bus cycle, the $\overline{HOLD}$ signal must remain asserted until state T4 of the DMAC cycle.

2.0 Architectural Description (Continued)

BIT-aligned Word Transfer

Syntax: BITWT

Setup: R0 Base address, source word
R1 Base address, destination double word
R2 Shift value

The BITWT instruction performs a fast logical OR operation between a source word and a destination double word, stores the result into the destination double word and increments registers R0 and R1 by two. Before performing the OR operation, the source word is shifted left (i.e., in the direction of increasing bit numbers) by the value in register R2.

This instruction can be used within the inner loop of a block OR operation. Its use assumes that the source data is 'clean' and does not need masking. The BITWT format is shown in *Figure 2-14.*

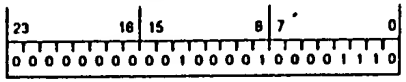


FIGURE 2-14. BITWT Instruction Format

External BitBLT

Syntax: EXTBLT

Setup:

R0	base addresses, source data
R1	base address, destination data
R2	width (in bytes)
R3	height (in lines)
R4	horizontal increment/ decrement
R5	temporary register (current width)
R6	source warp (adjusted)
R7	destination warp (adjusted)

Note 1: R0 and R1 are updated after execution to point to the last source and destination addresses plus related warps. R2, R3 and R5 will be modified. R4, R6 and R7 are returned unchanged

Note 2: Source and destination pointers should point to word-aligned operands to maximize speed and minimize external interface logic.

This instruction performs an entire BitBLT operation in conjunction with an external BitBLT Processing Unit (BPU). The external BPU Control Register should be loaded by the software before the instruction is executed (refer to the DP8510 or DP8511 data sheets for more information on the BPU). The NS32FX160 generates a series of source read, destination read and destination write bus cycles until the entire data block has been transferred. The BitBLT operation can be performed in either horizontal direction, as controlled by the sign of the contents of register R4.

Depending on the relative alignment of the source and destination blocks, an extra source read may be required at the beginning of each scan line to load the pipeline register in the external BPU. The L bit in the PSR register determines whether the extra source read is performed. If L is 1, no extra read is performed. The instructions CMPQB 2,1 or CMPQB 1,2 could be executed to provide the right setting for the L bit just before executing EXTBLT. *Figure 2-15* shows the EXTBLT format. The bus activity for a simple BitBLT operation is shown in *Figure 2-20.*

FIGURE 2-15. EXTBLT Instruction Format

2.4.3.2 Pattern Fill

Only one instruction is in this group. It is usually used for clearing RAM and drawing patterns and lines.

Move Multiple Pattern

Syntax: MOVMPI

Setup:

R0	base address of the destination
R1	pointer increment (in bytes)
R2	number of pattern moves
R3	source pattern

Note: R1 and R3 are not modified by the instruction. R2 will always be returned as zero. R0 is modified to reflect the last address into which a pattern was written.

This instruction stores the pattern in register R3 into the destination area whose address is in register R0. The pattern count is specified in register R2. After each store operation the destination address is changed by the contents of register R1. This allows the pattern to be stored in rows, in columns, and in any direction, depending on the value and sign of R1. The MOVMPI instruction format is shown in *Figure 2-16.*

FIGURE 2-16. MOVMPI Instruction Format

2.4.3.3 Data Compression, Expansion and Magnify

The three instructions in this group can be used to compress data and restore data from compression. A compressed character set may require from 30% to 50% less memory space for its storage.

The possible compression ratio can be 50:1 or higher depending on the data and algorithm used. TBITS can also be used to find boundaries of an object. As a character is needed, the data is expanded and stored in a RAM buffer. The expand instructions (SBITS, SBITPS) can also function as line drawing instructions.

Test Bit String

Syntax: TBITS option

Setup:

R0	base address, source (byte address)
R1	starting source bit offset
R2	destination run length limited code
R3	maximum value run length limit
R4	maximum source bit offset

Option:

1	count set bits until a clear bit is found
0	count clear bits until a set bit is found

Note: R0, R3 and R4 are not modified by the instruction execution. R1 reflects the new bit offset. R2 holds the result

This instruction starts at the base address, adds a bit offset, and tests the bit for clear if "option" = 0 (and for set if "option" = 1). If clear (or set), the instruction increments to the next higher bit and tests for clear (or set). This testing for clear proceeds through memory until a set bit is found or until the maximum source bit offset or maximum run length value is reached. The total number of clear bits is stored in the destination as a run length value.

When TBITS finds a set bit and terminates, the bit offset is adjusted to reflect the current bit address. Offset is then ready for the next TBITS instruction with "option" = 0. After the instruction is executed, the F flag is set to the value of the bit previous to the bit currently being pointed to (i.e., the value of the bit on which the instruction completed execution). In the case of a starting bit offset exceeding the maximum bit offset (R1 ≥ R4), the F flag is set if the option was 1 and clear if the option was 0. The L flag is set when the desired bit is found, or if the run length equalled the maximum run length value and the bit was not found. It is cleared otherwise. *Figure 2-17* shows the TBITS instruction format.

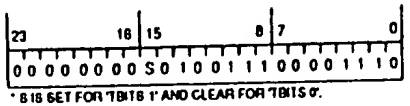


FIGURE 2-17. TBITS Instruction Format

Set Bit String

Syntax: SBITS

Setup: R0 base address of the destination
R1 starting bit offset (signed)
R2 number of bits to set (unsigned)
R3 address of string look-up table

Note: When the instruction terminates, the registers are returned unchanged.

SBITS sets a number of contiguous bits in memory to 1, and is typically used for data expansion operations. The instruction draws the number of ones specified by the value in R2, starting at the bit address provided by registers R0 and R1. In order to maximize speed and allow drawing of patterned lines, an external 1 Kbyte lookup table is used. The lookup table is specified in the NS32CG16 Printer/Display Processor Programmer's Reference Supplement.

When SBITS begins executing, it compares the value in R2 with 25. If the value in R2 is less than or equal to 25, the F flag is cleared and the appropriate number of bits are set in memory. If R2 is greater than 25, the F flag is set and no other action is performed. This allows the software to use a faster algorithm to set longer strings of bits. *Figure 2-18* shows the SBITS instruction format.

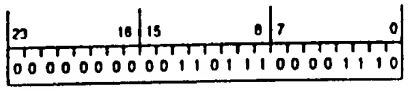


FIGURE 2-18. SBITS Instruction Format

Set BIT Perpendicular String

Syntax: SBITPS

Setup: R0 base address, destination (byte address)
R1 starting bit offset
R2 number of bits to set
R3 destination warp (signed value, in bits)

Note: When the instruction terminates, the R0 and R3 registers are returned unchanged. R1 becomes the final bit offset. R2 is zero.

The SBITPS can be used to set a string of bits in any direction. This allows a font to be expanded with a 90 or 270 degree rotation, as may be required in a printer application. SBITPS sets a string of bits starting at the bit address specified in registers R0 and R1. The number of bits in the string is specified in R2. After the first bit is set, the destination warp is added to the bit address and the next bit is set. The process is repeated until all the bits have been set. A negative raster warp offset value leads to a 90 degree rotation. A positive raster warp value leads to a 270 degree rotation. If the R3 value is = (space warp + 1 or -1), then the result is a 45 degree line. If the R3 value is +1 or -1, a horizontal line results.

SBITS and SBITPS allow expansion on any 90 degree angle, giving portrait, landscape and mirror images from one font. *Figure 2-19* shows the SBITPS instruction format.

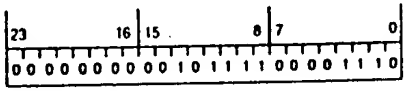


FIGURE 2-19. SBITPS Instruction Format

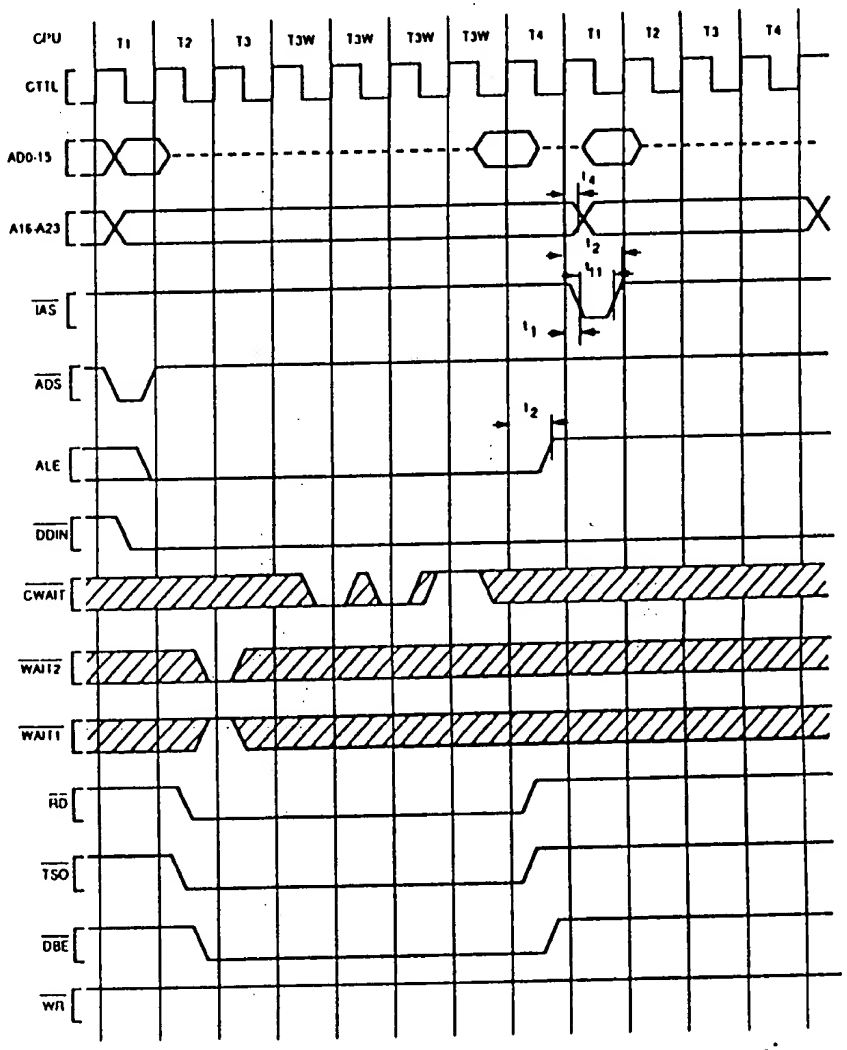


FIGURE 4-6. Off-Chip and On-Chip Read Cycles

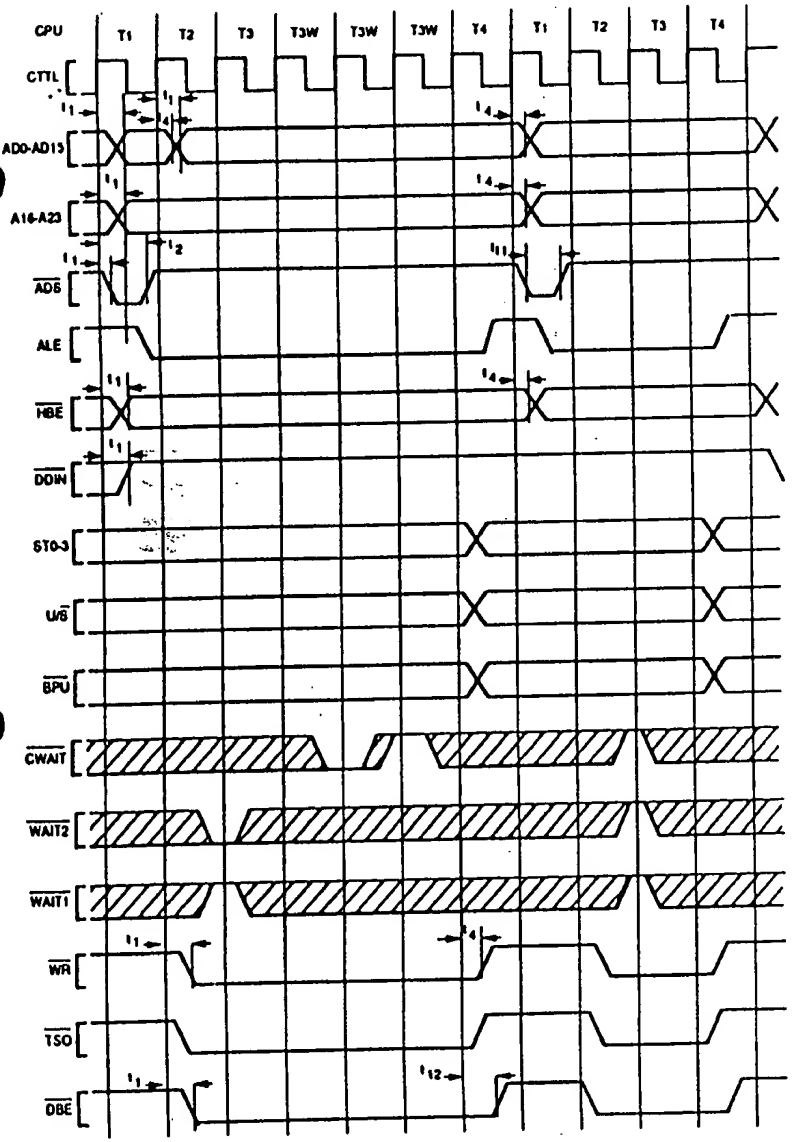


FIGURE 4-5. Write Cycle

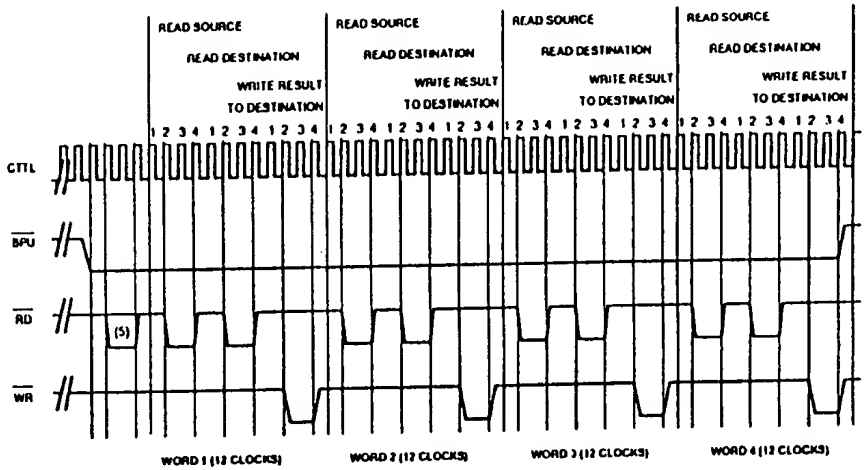


FIGURE 2-20. Bus Activity for a Simple BitBLT Operation

Note 1: The example is for a block 4 words wide and 1 line high.
Note 2: The sequence is common with all logical operations of the DP8510/DP8511 BPU.
Note 3: Mask values, shift values and number of bit planes do not affect the performance.
Note 4: Zero wait states are assumed throughout the BitBLT operation.
Note 5: The extra read is performed when the BPU pipeline register needs to be preloaded.

2.4.3.3.1 Magnifying Compressed Data

Restoring data is just one application of the SBITS and SBITPS instructions. Multiplying the "length" operand used by the SBITS and SBITPS instructions causes the resulting pattern to be wider, or a multiple of "length".

As the pattern of data is expanded, it can be magnified by 2x, 3x, 4x,..., 10x and so on. This creates several sizes of the same style of character, or changes the size of a logo. A magnify in both dimensions X and Y can be accomplished by drawing a single line, then using the MOVS (Move String) or the BB instructions to duplicate the line, maintaining an equal aspect ratio.

More information on this subject is provided in the NS32CG16 Printer/Display Processor Programmer's Reference Supplement.

2.5 FAX ACCELERATOR MODULE

The FAX Accelerator Module (FAM) performs arithmetic operations on vectors of complex numbers. High performance is achieved by using a Hardware Multiplier Accumulator, an Address Generator for main memory operand accesses, and an on-chip RAM array.

The FAM executes complex arithmetic calculations on two vector operands. One vector is stored in the internal RAM array. The other vector is either organized as a circular buffer in the main memory or stored in the internal RAM array.

The FAM executes vector operations in a two stage pipeline. This allows for a significant performance enhancement as each operand fetch and execution on different vector elements are performed simultaneously rather than in a strictly sequential manner. The FAM can fetch up to two data elements at a time, using its address generator. The first operand is fetched from the coefficient array whereas the second operand is from either external memory or from the coefficient array. While fetching operands for one vector element, the FAM performs the multiply and add operations on the previous vector element. Each complex multiply and accumulate operation requires two operand fetches, four multiplications and four additions. The FAM pipeline allows a maximal throughput of a complex multiply-accumulate operation in 8 clock cycles.

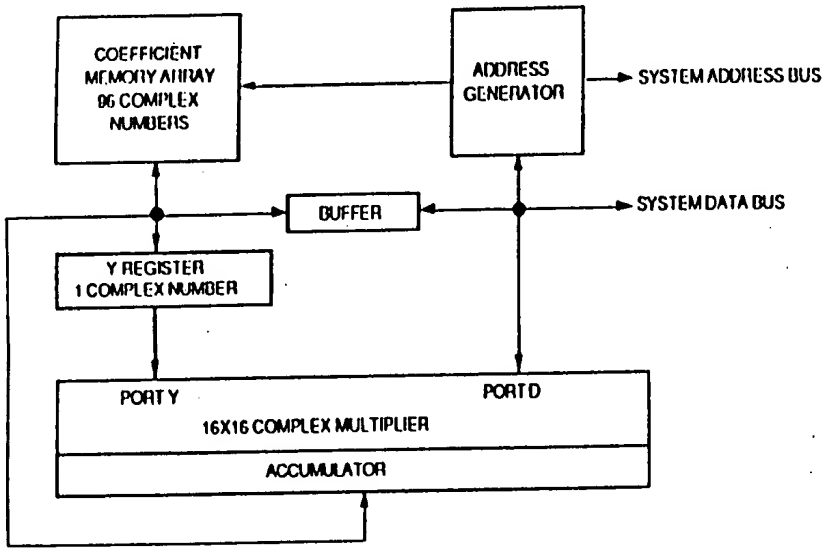


FIGURE 2.21. Fax Accelerator Module Block Diagram

2.5.1 FAM Operation

The following terms are used for the description of operations:

C[i]	Coefficient memory element, entry [i] can be selected by address generator or directly accessed by CPU.
D[i]	Data from external memory fetched using the address generator.
Y	Complex Multiplier input register.
D[i]*	The conjugate of D[i].
A	Complex Accumulator.

The FAX Accelerator Module can execute 6 basic commands:

VCMAC	Vector Complex Multiply Accumulate
VCMAG	Vector Complex Magnitude
VCMAD	Vector Complex Multiply Add
VCMUL	Vector Complex Multiply
LOAD	Write into C,Y,A, or CTL
STORE	Read from C, A, Y, ST or CTL

VCMAC, VCMAD and VCMUL commands use the following parameters:

D vector starting address
C vector starting address
Vector length
Control bits

VCMAG command uses only the last three operands.

2.5.1.1 Complex Number Representation

Complex numbers are organized as double words. Each double word contains two 16-bit 2's complement fractional integers. The less significant word contains the Real part of the number. The most significant word contains the Imaginary part of the number.

Complex vectors consist of arrays of complex numbers stored in consecutive addresses. Complex vectors MUST be aligned to double word boundary. *Figure 2-22* illustrates the memory organization of vector D.

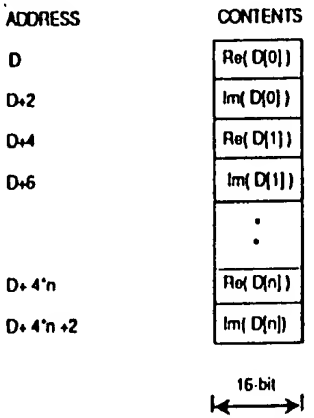


FIGURE 2-22. Memory Organization of a Complex Vector

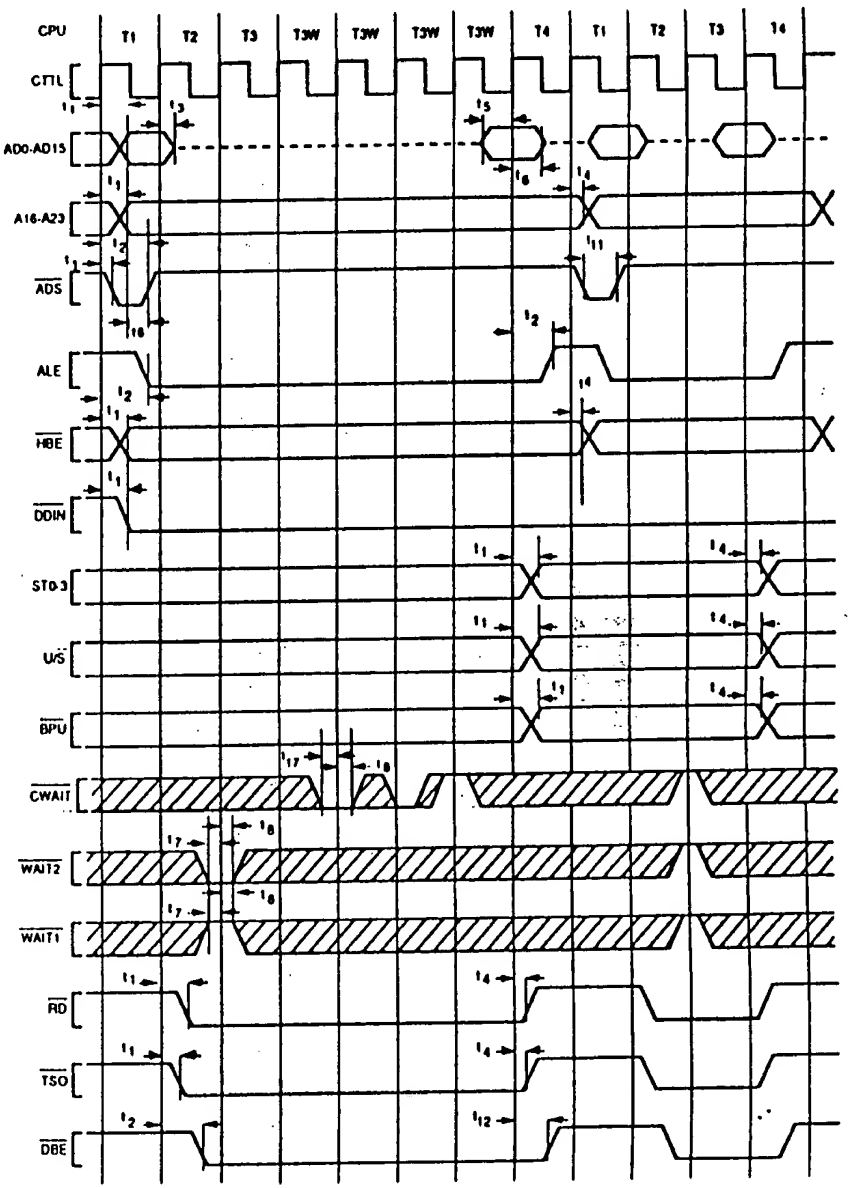


FIGURE 4-4. Read Cycle

4.0 Device Specifications (Continued)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32FX16-15, NS32FX16-20 and NS32FX16-25

Capacitive Load: CTTL -100pF, all other outputs -50pF

Symbol	Parameter	Reference	25 MHz		20 MHz		15 MHz		Units
			Min	Max	Min	Max	Min	Max	
t1	Output valid time	RE CTTL		12		13		14	ns
t[redacted]	Output valid time	RE CTTL		.5t26 +14		5t26 +15		5t26 +16	ns
t3	Output float time	RE CTTL	0	12	0	13	0	14	ns
t4	Output hold time	RE CTTL	0		0		0		ns
t5	Input setup time	RE CTTL	10		14		15		ns
t6	Input hold time	RE CTTL	2		2		2		ns
t7	Input setup time	RE CTTL	20		21		22		ns
t8	Input hold time	RE CTTL	2		2		2		ns
t9	Input setup time	RE CTTL	14		15		16		ns
t10	Input hold time	RE CTTL	2		2		2		ns
t11	Pulse width	0.8V	10		15		20		ns
t12	Output hold time	RE CTTL	.5t26 -2		.5t26 -2		.5t26 -2		ns
t13	Input setup time	FE CTTL	12		14		15		ns
t14	Input hold time	FE CTTL	2		2		2		ns
t15	Input hold time	RE CTTL	2		2		2		ns
t16	Output valid to strobe inactive		10		10		10		ns
t17	Input setup time	RE CTTL	10		18		22		ns
t20	OSCIN period	RE OSCIN	20	500	25	500	33	500	ns
t21	OSCIN high time	4.2V	.5t20 -3		.5t20 -4		.5t20 -5		ns
t22(1)	OSCIN low time	1.0V	.5t20 -3		.5t20 -4		.5t20 -5		ns
t[redacted]	OSCIN-CTTL delay	4.2V RE OSCIN to RE CTTL		25		29		35	ns
t26	CTTL period	2.0V	40	1000	50	1000	66	1000	ns
t27	CTTL high time	2.0V	.5t26 -4		.5t26 -5		.5t26 -6		ns
t28	CTTL low time	0.8V	.5t26 -4		.5t26 -5		.5t26 -6		ns
t29	CTTL fall time	RE CTTL		4		5		6	ns
t30	CTTL rise time	FE CTTL		4		5		6	ns
t38	INT signal hold	after interrupt acknowledge		8		8		8	CTTL period
t32	OSCIN to FCLK RE delay	4.2V RE OSCIN to RE FCLK		15		20		25	ns
t33	FCLK to CTTL RE delay	RE FCLK to RE CTTL		10		10		10	ns
t34	FCLK to CTTL FE delay	RE FCLK to FE CTTL		10		10		10	ns

1. Not 100% tested

2.0 Architectural Description (Continued)

2.5.1.2 Mac Operation

The ALU of the FAX Accelerator Module contains a 16*16 multiplier and a 32-bit addor. Bits 15-30 (16 bits) of the result are rounded, and can be read by accessing the A register. If an overflow is detected during operation, the ST register OVF bit and either OP0 or OP1 bits will be set to "1".

A 16-bit value is loaded into bits 15-30 of the Accumulator and the lower bits are set to "0". The value from bit 30 is copied into bit 31 for sign extension. Bit 14 is set to "1". An overflow is detected whenever the value of bit 30 is different from the value of bit 31.

2.5.1.3 Instruction Set

Each instruction of the FAM is controlled by two op-code bits (OPC0 and OPC1), and two specifiers, COJ and CLR. COJ specifies whether or not the operand on port D of the multiplier needs to be conjugated prior to multiplication. The CLR bit is used to extend the instruction set. On VCMAC and VCMAG, CLR specifies whether or not the Accumulator has to be cleared at the beginning of the vector operation. On VCMAD, CLR is set to specify that the operation will ignore the value of C[i]. In VCMUL, CLR is set to indicate that the value of D[i] is to be taken, instead of 1+D[i]. Table 2.4 is a summary of the various instruction sequences executed by the FAX Accelerator Module as a function of OPC1, OPC0, COJ, and CLR bits in the CTL register.

The summation is done on N elements of the vector.

All operands are complex numbers. Thus,
$A=\sum (C[i] \times D[i])$ breaks down to:
$Re(A) = \sum \{Re(C[i]) \times Re(D[i]) - Im(C[i]) \times Im(D[i])\}$
$Im(A) = \sum \{Re(C[i]) \times Im(D[i]) + Im(C[i]) \times Re(D[i])\}$

Note that the Accumulator (A), the multiplier input register (Y), the external data pointer (DPTR) and the coefficient pointer (CPTR) registers are used as temporary registers during vector operations. The values previously stored in those registers are destroyed. If the contents of the Accumulator (A) register after a FAM operation is used as an initial value for the next FAM operation, it should be noted that the least significant bits of A (0+14) may contain a value other than zero.

2.5.1.4 Circular Buffers

The FAM accesses arrays of data in external memory using the DPTR as an address pointer. DS0 and DS1 bits of the CTL register control the size of the array. The FAM allows a convenient way of handling the data array as in a FIFO. Only the appropriate number of the least significant bits of the DPTR are incremented on each access. The upper bits remain constant. Table 2.5 shows which bits are incremented. The rest remain constant.

Instruction	OPC1	OPC0	CLR	COJ	Operation
VCMAD	0	0	0	0	C[i] <= C[i] + Y x D[i]
	0	0	0	1	C[i] <= C[i] + Y x D[i]*
	0	0	1	0	C[i] <= Y x D[i]
	0	0	1	1	C[i] <= Y x D[i]*
VCMUL	0	1	0	0	C[i] <= C[i] x (1 + D[i])
	0	1	0	1	C[i] <= C[i] x (1 + D[i]*)
	0	1	1	0	C[i] <= C[i] x D[i]
	0	1	1	1	C[i] <= C[i] x D[i]*
VCMAC	1	0	0	0	A <= A + Σ (C[i] x D[i])
	1	0	0	1	A <= A + Σ (C[i] x D[i]*)
	1	0	1	0	A <= Σ (C[i] x D[i])
	1	0	1	1	A <= Σ (C[i] x D[i]*)
VCMAG	1	1	0	0	A <= A +Σ (C[i] x C[i])
	1	1	0	1	A <= A + Σ (C[i] xC[i]*)
	1	1	1	0	A <= Σ (C[i] x C[i])
	1	1	1	1	A <= Σ (C[i] x C[i]*)

TABLE 2.4. FAX Accelerator Instruction Set Summary

2.5.1.5 Performance Considerations

The FAX Accelerator Module is designed for optimal throughput in vector operations. Its two stage pipeline overlaps the execution of operand fetches and multiply-accumulate operations for different vector elements. The FAM can fetch up to two data elements at a time, using its address generator for main memory access and the coefficient array for the second operand. While fetching operands for one vector element, the FAM performs the multiplication and additions on the previous vector element. Each complex multiply and accumulate operation requires two operand fetches, four multiplications and four additions. The FAM pipeline allows a maximal throughput of a complex multiply accumulate operation in 8 clock cycles. See Section B4, FAX Accelerator Module Performance, for more details.

Access to the FAM registers while it is executing a vector operation are delayed (as if the CWAIT input is active). When the FAM finishes the operation, access to the registers proceeds.

The FAM uses the full bandwidth of the external bus during VCMAD, VCMUL or VCMAC operations. While executing the VCMAG instruction, the bus is free as no external operands are required. In this case the core CPU proceeds execution in parallel with the FAM operation.

DS1	DS0	External Buffer Size(DW)	Constant Address bits	Incremented Address bits
0	0	8	A0, A5-A23	A1-A4
0	1	16	A0, A6-A23	A1-A5
1	0	32	A0, A7-A23	A1-A6
1	1	64	A0, A8-A23	A1-A7

TABLE 2.5. Circular Buffer Size

During VCMAD, VCMUL or VCMAC operations, external HOLD requests will be granted at the end of each memory access. Note that interrupt requests cannot be acknowledged until the FAM finishes a vector operation.

2.5.2 FAM Registers and RAM Array

The FAM contains 7 registers and a 96 double-word RAM array. These registers and internal RAM can be accessed as memory-mapped I/O devices. Any reference to the registers and the RAM is done using the on-chip bus protocol. See Section 3.4.7.

All the registers, except for the Status Register (ST), are readable and writeable. ST is read-only. Accessing the register memory locations should be a multiple of a byte-length. Word accesses must be on word boundary, and double-word accesses must be on double word boundary. Failing to do so will cause unpredictable results.

2.5.2.1 Coefficient RAM Array C[0]-C[95]

Each register in the coefficient array is 32-bits wide and holds one complex number. See Section 2.5.1.1.

Note that the RAM array is not limited to coefficient storage only. It can be used as a fast, zero-wait state on-chip memory for instructions and data storage.

However, the RAM can be used for instruction storage only if the instructions are loaded into this RAM using word-aligned accesses. This can be achieved by moving aligned double words from the external memory to the on-chip RAM. Data can also be stored in the on-chip RAM with one restriction: storing data in the on-chip RAM can be done only if all the data is written using aligned word or double-word accesses.

2.5.2.2 Multiplier Input Register Y

This 32-bit register holds one complex operand (see Section 2.5.1.1). The Y register is mapped into two consecutive words called Y0 and Y1.

2.5.2.3 Accumulator A

This 32-bit register holds one complex result (see Sec. 2.5.1.1). The A register is mapped into two consecutive words, also called A0 and A1.

Internally, A0 and A1 are 32-bit registers, however, only bits 15-30 (16 bits) are accessible. The rest of the bits are used for a higher dynamic range on intermediate calculations.

2.5.2.4 Data Pointer DPTR

This is a 24-bit pointer to the beginning of the data vector in the main memory. In order to implement circular buffers, only the least significant bits of the DPTR pointer are incremented. When the end of a buffer is reached, the least significant bits of DPTR are reloaded with zeros. The number of bits that are set to zero (which defines the size of the circular buffer) is controlled by CTL. The least-significant word of the DPTR is called DPTR0, and the most-significant byte is called DPTR1.

2.5.2.5 Coefficient Memory Vector Pointer CPTR.

The CPTR register holds the address and length of the coefficient vector.

15							8	7							0
0	L6	L5	L4	L3	L2	L1	L0	0	S6	S5	S4	S3	S2	S1	S0

S0-S6 Start address of coefficient's vector (number of C reg).

L0-L6 Length of coefficient's vector (in double words).

Specifying 0 as the value of CPTR.L will cause an unpredictable result.

2.5.2.6 Control Register CTL

The CTL register controls the various modes of operation. For more details see Section 2.5.1.

7							0
OPC1	OPC0	DS1	DS0	X	X	CLR	COJ

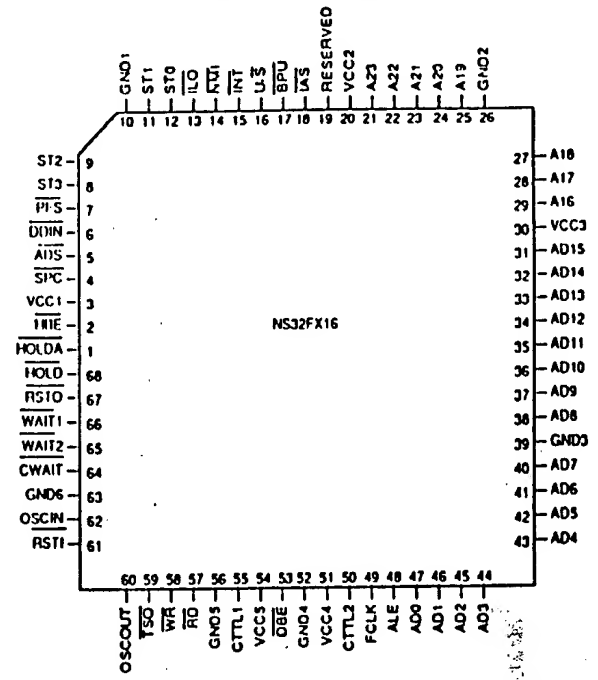
OPC1-0 Operation code.

00	VCMAD	Vector Complex Multiply Add
01	VCMUL	Vector Complex Multiply
10	VCMAC	Vector Complex Multiply Accumulate
11	VCMAG	Vector Complex Magnitude

Register	Address	Length (bytes)	Direction
C[0]-C[95]	FFFFD000 - FFFFD17F	96 x 4 = 384	R/W
reserved	FFFFD180 - FFFFD3FF	640	---
A	FFFFD400	4	R/W
Y	FFFFD404	4	R/W
DPTR	FFFFD408	3	R/W
CPTR	FFFFD40c	2	R/W
CTL	FFFFD40e	1	R/W
ST	FFFFD410	1	R

TABLE 2.6 FAM Register Address Map

68-Pin PCC Package



Bottom View

FIGURE 4-1. Connection Diagram

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 0.8V or 2.0V on the rising or falling edges of CTTL when the capacitive loading of CTTL is 100 pF, unless specifically stated otherwise. The timing specifications refer to 0.8 or 2.0V on the TTL output and input signals as illustrated in *Figures 4-2* and *4-3*, unless specifically stated otherwise.

ABBREVIATIONS:

L.E.—leading edge R.E.—rising edge
T.E.—trailing edge F.E.—falling edge

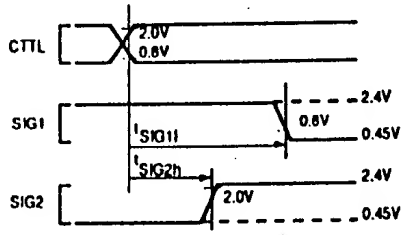


FIGURE 4-2. Timing Specifications Standard (Signal Valid After Clock Edge)

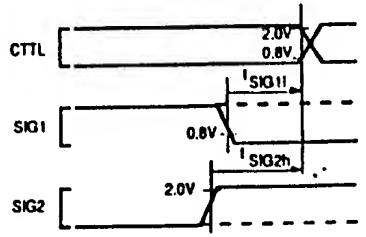


FIGURE 4-3. Timing Specification Standard (Signal Valid Before Clock Edge)

4.0 Device Specifications (Continued)

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.

Temperature Under Bias	0°C to 70°C
Storage Temperature	−65°C to +150°C
All input or Output Voltages with Respect to GND	−0.5V to +7V

Note: *Applying voltage beyond that level might overstress the part. Applying voltage on logic pins beyond that level might cause latchup to the product.*

4.3 ELECTRICAL CHARACTERISTICS: T_A = 0°C to +70°C, V_{CC} = 5V ±10%, GD = 0V

Symbol	Parameter	Test Conditions	Min	Typ	Max	Unit
VIH	High Level Input Voltage		2.0		V_{CC} +0.5	V
VIL	Low Level Input Voltage		−0.5		0.8	V
VXL	OSCIN Input Low Voltage				0.5	V
VXH	OSCIN Input High Voltage		4.5			V
VOH	High Level Output Voltage	I_{OH} = −400 µA	2.4			V
VOL	Low Level Output Voltage	I_{OL} = 4 mA			0.45	V
II	Input Load Current	V_{CC} = 5.5V, V_{SS} = 0V, V_{IN} = 0, 5.5V(1)	−20		20	µA
IL	Leakage Current	V_{CC} = 5.5V, V_{SS} = 0V, V_{OUT} = 0.4, 5.5V	−20		20	µA
IILS	SPC Input Current (low)	V_{IN} = 0.4V, SPC In Input Mode				mA
ITTL	CTTL Input Current (low)				4.0	mA
ICC	Supply Current	25 MHz, T_A = 25°C, Iout = 0(2)		170	240	mA

Note 1: For all inputs, except $\overline{SPC}$.

Note 2: I_{CC} is affected by the clock scaling factor selected by the C and M bits in the CFG register; see Section 3.2.1.

2.0 Architectural Description (Continued)

DS0-DS1 Data Buffer Size.

00	8 double-words
01	16 double-words
10	32 double-words
11	64 double-words

CLR Clear Accumulator (A0 and A1) when set to 1 prior to beginning the operation. For more details, see section 2.5.1.3.

COJ Conjugate when set to 1. The value of the operand in port D of the multiplier will be conjugated prior to multiplication.

2.5.2.7 Status Register ST

The ST register holds the status of the last vector operation.

7							0
OVF	X	X	X	X	X	OP1	OP0

OVF Overflow Indication (see section 2.5.1.2).

OP0 Overflow occurred on calculation of A0.

OP1 Overflow occurred on calculation of A1.

The ST register is cleared to 0 in the following cases:

- the user writes directly to either A0 or A1,
- the user writes to the CTL register,
- upon reset.

3.0 Functional Description

3.1 POWER AND GROUNDING

The NS32FX16 requires a single 5-Volt power supply applied on 5 pins: VCC1-VCC5.

Grounding connections are made on 6 pins: GND1-GND6.

For optimal noise immunity, the power and ground pins should be connected to V_{CC} and ground planes respectively. If V_{CC} and ground planes are not used, single conductors should be run directly from each V_{CC} pin to a power point, and from each GND pin to a ground point. Daisy chained connections should be avoided.

Decoupling capacitors should also be used to keep the noise level to a minimum. Two standard 0.1 µF ceramic capacitors can be used for this purpose. In addition, a 1.0 µF tantalum capacitor should be connected between V_{CC} and ground. They should be attached to V_{CC}, V_{SS} pairs as close as possible to the NS32FX16.

3.2 CLOCKING

The NS32FX16 provides an internal oscillator that interacts with an external clock source through two signals: OSCIN and OSCOUT.

Either an external signal-phase clock signal or a crystal can be used as the clock source. If a single-phase clock source is used, only the connection on OSCIN is required; OSCOUT should be left unconnected or loaded with no more than 5 pF of stray capacitance. The voltage level requirements specified in Section 4.3 must also be met for proper operation.

When operation with a crystal is desired, special care should be taken to minimize stray capacitances and inductances. The crystal, as well as the external RC components, should be placed in close proximity to the OSCIN and OSCOUT pins to keep the printed circuit trace lengths to an absolute minimum.

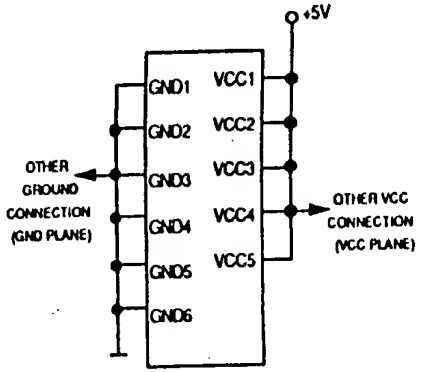


FIGURE 3-1. Power and Ground Connections

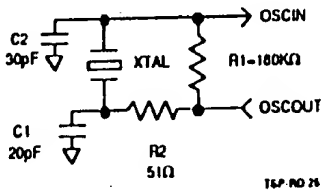


FIGURE 3-2(a). Crystal Interconnections - 30 MHz

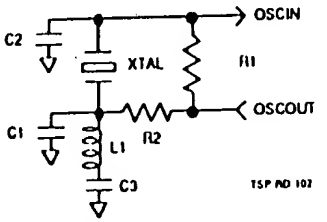


FIGURE 3-2(b). Crystal Interconnections - 40 MHz, 50 MHz

RCL Component Value

Frequency (MHz)	R1 (kΩ)	C1 (pF)	C2 (pF)	C3 (pF)	L1 (µH)	R2 (Ω)
40	150	20	20	200	1	51
50	150	20	20	200	0.6	51

TABLE 3-1. External Oscillator Specifications

Crystal Characteristics

Type.....At-Cut
Tolerance.....0.005% at 25°C
Stability.....0.01% from 0°C to 70°C
Resonance.....30MHz=Fundamental (parallel)
.....40, 50 MHz - Third Overtone (parallel)
Maximum Shunt Capacitance.....7pF
Maximum Series Resistance.....50Ω

3.2.1 Power Save Mode

The NS32FX16 provides a power save feature that can be used to significantly reduce the power consumption at times when the computational demand decreases. The device uses the clock signal at the OSCIN pin to derive the internal clock as well as the external signals CTTL and FCLK. The frequency is affected by the clock scaling factor. Scaling factors of 1, 2, 4 or 8 can be selected by properly setting the C and M bits in the CFG register. The power save mode should not be used to reduce the CTTL clock frequency below the minimum frequency required by the CPU (1MHz).

Upon reset, both C and M are set to zero; thus, maximum clock rate is selected.

Due to the fact that the C and M bits are programmed by the SETCFG instruction, the power save feature can only be controlled by programs running in supervisor mode.

The following table shows the C and M bit settings for the various scaling factors, and the resulting supply current for a crystal frequency of 50 MHz.

Clock Scaling Factor vs Supply Current

C	M	Scaling Factor	CPU Clock Frequency	Typical I_{CC} at +5V
0	0	1	25 MHz	170 mA
0	1	2	12.5 MHz	91 mA
1	0	4	6.25 MHz	50 mA
1	1	8	3.13 MHz	30 mA

3.3 RESETTING

The $\overline{RSTI}$ input pin is used to reset the NS32FX16. The CPU samples $\overline{RSTI}$ on the falling edge of CTTL.

Whenever a low level is detected, the CPU responds immediately. Any instruction being executed is terminated; any results that have not yet been written to memory are discarded; and any pending interrupts and traps are eliminated. The internal latch for the edge-sensitive $\overline{NMI}$ signal is cleared.

On application of power $\overline{RSTI}$ must be held low for at least 50 µs after VCC is stable. This is to ensure that all on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must also remain active for not less than 64 CTTL cycles. See *Figures 3-4* and *3-5*.

While in the Reset state, the CPU drives the signals $\overline{ADS}$, $\overline{RD}$, $\overline{WR}$, $\overline{DBE}$, $\overline{TSO}$, $\overline{BPU}$, and $\overline{DDIN}$ inactive. AD0-AD15, A16-A23 and $\overline{SPC}$ are floated, and the state of all other output signals is undefined.

The internal CPU clock and CTTL run at half the frequency of the signal on the OSCIN pin. FCLK runs at the same frequency as OSCIN.

The $\overline{HOLD}$ signal must be kept inactive. After the $\overline{RSTI}$ signal is driven high, the CPU will stay in the reset condition for approximately 8 clock cycles and then it will begin execution at address 0.

The PSR is reset to 0. The CFG C and M bits are reset to 0. $\overline{NMI}$ is enabled to allow Non-Maskable Interrupts. The following conditions are present after reset due to the PSR being reset to 0:
Tracing is disabled.
Supervisor mode is enabled.
Supervisor stack space is used when the TOS addressing mode is indicated.
No trace traps are pending.
Only $\overline{NMI}$ is enabled. $\overline{INT}$ is not enabled.
$\overline{BPU}$ is inactive high.
The Clock Scaling Factor is set to 1; refer to Section 3.2.1.

Note that vector/non-vectored interrupts have not been selected. While interrupts are disabled, a SETCFG [I] instruction must be executed to declare the presence of the NS32202 if vectored interrupts are desired. If non-vectored interrupts are required, a SETCFG without the [I] must be executed.

The presence/absence of the NS32081 or NS32381 has also not been declared. If there is a Floating-Point Unit, a SETCFG [F] instruction must be executed. If there is no floating-point unit, a SETCFG without the [F] must be executed.

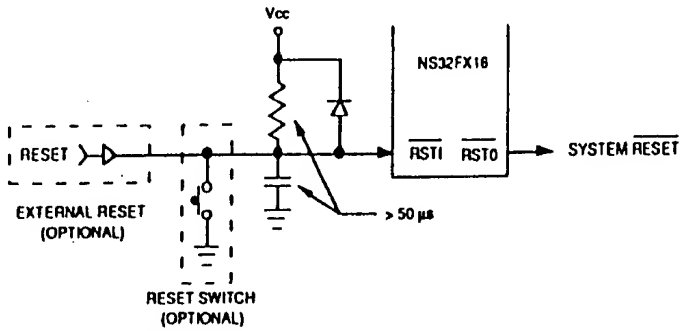


FIGURE 3-3. Recommended Reset Connections

ST0-ST3 **Bus Status**
Bus cycle status code. ST0 is the least significant bit. Encodings are:

0000 — Idle: CPU inactive on bus.
0001 — Idle: WAIT Instruction.
0010 — FAM Data Transfer.
0011 — Idle: Waiting for Slave.
0100 — Interrupt Acknowledge Master.
0101 — Interrupt Acknowledge, Cascaded.
0110 — End of Interrupt, Master.
0111 — End of Interrupt, Cascaded.
1000 — Sequential Instruction Fetch.
1001 — Non-Sequential Instruction Fetch.
1010 — Data Transfer.
1011 — Read Read-Modify-Write Operand.
1100 — Read for Effective Address.
1101 — Transfer Slave Operand.
1110 — Read Slave Status Word.
1111 — Broadcast Slave ID.

$\overline{TSO}$ **Timing State Output.**
The falling edge of $\overline{TSO}$ identifies the beginning of state T2 of a bus cycle. The rising edge identifies the beginning of state T4.

U/$\overline{S}$ **User/Supervisor.**
User or Supervisor Mode status. High indicates User Mode; low indicates Supervisor Mode.

$\overline{WR}$ **Write Strobe.**
Activated during CPU or DMA write cycles to enable writing of data to memory or peripherals.

4.1.4 Input-Output Signals

AD0-AD15 **Address/Data Bus.**
Multiplexed Address/Data informati
Bit 0 is the least significant bit of each

$\overline{ADS}$ **Address Strobe.**
Signals the beginning of a bus cycle a
can be used for controlling the addr
latches. During $\overline{HLDA}$ asserted, t
signal becomes an input and the C
monitors it to detect the beginning of
external DMA cycle and generate
relevant strobe signals. When
external DMA Controller is used, A
should be pulled up to V_{CC} throug
10K resistor.

$\overline{DDIN}$ **Data Direction.**
Status signal indicating the direction
the data transfer during a bus cyc
During $\overline{HLDA}$ asserted, this sig
becomes an input and determines
activation of $\overline{RD}$ or $\overline{WR}$.

$\overline{SPC}$ **Slave Processor Control.**
Used by the CPU as the data stro
output for slave processor transf
used by a slave processor
acknowledge completion of a sla
instruction.

4.0 Device Specifications

4.1 PIN DESCRIPTIONS

4.1.1 Supplies

VCC1-5 Power
+5V positive supplies

GND1-6 Ground

4.1.2 Input Signals

$\overline{CWAIT}$ **Continuous Wait.**
Causes the CPU to insert continuous wait states if sampled low at the end of T2 and each following T3 or T3W state. $\overline{WAIT1}$-$\overline{WAIT2}$ inputs are sampled by the CPU during T3 or T3W if $\overline{CWAIT}$ asserted (low) and the corresponding wait-state counter is initialized. The wait states due to $\overline{WAIT1}$-$\overline{WAIT2}$ (if any) are added only after $\overline{CWAIT}$ is removed (becomes high). See Section 3.4.3.

$\overline{HOLD}$ **Hold Request.**
When active, causes the CPU to release the bus for DMA or multiprocessing purposes. See Section 3.5.

Note: If the $\overline{HOLD}$ signal is generated asynchronously, its setup and hold times may be violated. In this case, it is recommended to synchronize it with CTTL to minimize the possibility of metastable states.

The CPU provides only one synchronization stage to minimize the $\overline{HLDA}$ latency. This is to avoid speed degradations. In case of heavy $\overline{HOLD}$ activity (i.e., DMA controller cycles interleaved with CPU cycles).

$\overline{INT}$ **Interrupt.**
A low level on this pin requests a maskable interrupt, $\overline{INT}$ must be asserted until the interrupt is acknowledged.

Non-Maskable Interrupt.
A High-to-Low transition on this signal requests a non-maskable interrupt.

OSCIN **Crystal/External Clock Input.**
Input from a crystal or an external clock source.

$\overline{RSTI}$ **Reset Input.**
Asynchronous signal (Schmitt triggered) used to generate a CPU reset.

$\overline{WAIT1}$ **Wait State Inputs.**

$\overline{WAIT2}$ Binary weighted inputs, allowing from zero to three wait states to be specified. The $\overline{WAIT1}$-$\overline{WAIT2}$ value is sampled by the CPU at the end of T2 (if $\overline{CWAIT}$ not asserted) or at the end of the last T3 or T3W state in which $\overline{CWAIT}$ is asserted. See Section 3.4.3.

4.1.3 Output Signals

A16-A23 **High-Order Address Bits.**
These are the most significant 8 bits of the memory address bus.

ALE **Address Latch Enable.**
Controls address latches.

$\overline{BPU}$ **BPU Cycle.**
Activated (low) during a bus cycle to enable an external BitBLT processing unit. The EXTBLT instruction activates this signal.

CTTL1-2 **System Clock.**
CTTL1 and CTTL2 should be connected together externally.

$\overline{DBE}$ **Data Buffers Enable.**
Used to control external data buffers. It is active when the data buffers are to be enabled.

FCLK **Fast Clock.**
This clock is derived from the clock waveform on OSCIN. Its frequency is either the same as OSCIN or is lower, depending upon the scale factor programmed into the CFG register. See Section 3.2.1.

$\overline{HBE}$ **High Byte Enable.**
Status signal used to enable data transfers on the most significant byte of the data bus.

$\overline{HLDA}$ **Hold Acknowledge.**
Activated by the CPU in response to the $\overline{HOLD}$ input to indicate that the CPU has released the bus.

$\overline{IAS}$ **Internal Address Strobe.**
Signals the beginning of an on-chip bus cycle. $\overline{IAS}$ is a status signal used for debugging and tracing.

$\overline{ILO}$ **Interlocked Operation.**
When active (low), indicates that an interlocked operation is being executed.

OSCOUT **Crystal Output.**
This line is used as the return path for the crystal (if used). It must be left open when an external clock source is used to drive OSCIN.

$\overline{PFS}$ **Program Flow Status.**
A pulse on this line indicates the beginning of execution of an instruction.

$\overline{RD}$ **Read Strobe.**
Activated during CPU or DMA read cycles to enable reading of data from memory or peripherals.

$\overline{RSTO}$ **Reset Output.**
Asserted (low) when $\overline{RSTI}$ is low, initiating a system reset.

3.0 Functional Description (Continued)

In general, a SETCFG instruction must be executed in the reset routine, in order to properly configure the CPU. The options should be combined and executed in a single instruction. For example, to declare vectored interrupts, a Floating-Point unit installed, and full CPU clock rate, execute a SETCFG [F, I] instruction. To declare non-vectored interrupts, no FPU, and full CPU clock rate, execute a SETCFG [ ] instruction.

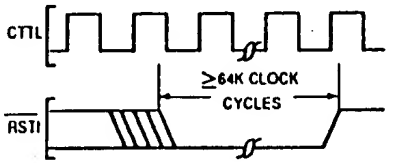


FIGURE 3-4. General Reset Timing

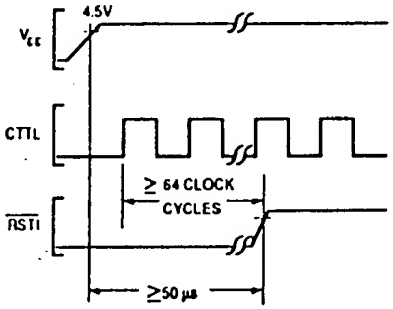


FIGURE 3-5. Power-on Reset Requirements

3.4 BUS CYCLES

The CPU will perform a bus cycle for one of the following reasons:

1) To write or read data, to or from memory or peripheral devices. Peripheral input and output are memory-mapped in National's Embedded System Processor family.
2) To fetch instructions into the eight-byte instruction queue. This happens whenever the bus would otherwise be idle and the queue is not already full.
3) To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
4) To transfer information to or from a Slave Processor.
5) To indicate an internal bus cycle (e.g., read of an on-chip FAM control register).

In terms of bus timing, cases 1 through 3 above are identical. For timing specifications, see Section 4. The only external difference between them is the four-bit code placed on the Bus Status pins (ST0-ST3).

Slave Processor cycles differ in that separate control signals are applied and there is no address involved (Section 3.4.9).

When using an external DMA channel, NS32FX16 still generates bus control signals if the DMA controller provides inputs that indicate the beginning of the DMA cycle ($\overline{ADS}$) and the cycle type ($\overline{DDIN}$). However, the address is generated in this case by the external DMA Controller.

Case 5 does not represent an active bus cycle ($\overline{ADS}$ not asserted; instead, a special address strobe, $\overline{IAS}$, is asserted). The purpose of these cycles is to allow a debug or trace device (e.g., ISE) to track bus transactions inside the chip.

3.4.1 Bus Status

The NS32FX16 CPU presents four bits of Bus Status information on pins ST0-ST3. The various combinations on these pins indicate why the CPU is performing a bus cycle, or, if it is idle on the bus, then why it is idle.

The Bus Status pins are interpreted as a four-bit value, with ST0 the least significant bit. Their values decode as follows:

0000 — The bus is idle because the CPU does not need to perform a bus access.

0001 — The bus is idle because the CPU is executing the WAIT instruction.

0010 — FAM Data Transfer.

0011 — The bus is idle because the CPU is waiting for a Slave Processor to complete an instruction.

0100 — Interrupt Acknowledge, Master.
The CPU is performing a Read cycle to acknowledge an interrupt request. See Section 3.4.6.

0101 — Interrupt Acknowledge, Cascaded.
The CPU is reading an interrupt vector to acknowledge a maskable interrupt request from a Cascaded Interrupt Control Unit.

0110 — End of Interrupt, Master.
The CPU is performing a Read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction at the completion of an interrupt's service procedure.

0111 — End of Interrupt, Cascaded.
The CPU is performing a read cycle from a Cascaded Interrupt Control Unit to indicate that it is executing a Return from Interrupt (RETI) instruction at the completion of an interrupt's service procedure.

1000 — Sequential Instruction Fetch.
The CPU is reading the next sequential word from the instruction stream into the Instruction Queue. It will do so whenever the bus would otherwise be idle and the queue is not already full.

1001 — Non-Sequential Instruction Fetch.
The CPU is performing the first fetch of instruction code after the Instruction Queue

is purged. This will occur as a result of any jump or branch, any interrupt or trap, or execution of certain instructions.

1010 — Data Transfer.
The CPU is reading or writing an operand of an instruction.

1011 — Read RMW Operand.
The CPU is reading an operand which will subsequently be modified and rewritten. The write cycle of RMW will have a "write" status.

1100 — Read for Effective Address Calculation.
The CPU is reading information from memory in order to determine the Effective Address of an operand. This will occur whenever an instruction uses the Memory Relative or External addressing mode.

1101 — Transfer Slave Processor Operand.
The CPU is either transferring an instruction operand to or from a Slave Processor, or it is issuing the Operation Word of a Slave Processor instruction. See Section 3.4.9.2.

1110 — Read Slave Processor Status.
The CPU is reading a Status Word from a Slave Processor after the Slave Processor has signalled completion of an instruction.

1111 — Broadcast Slave ID.
The CPU is initiating the execution of a Slave Processor Instruction by transferring the first byte of the instruction, which represents the slave processor identification.

3.8.2 Floating-Point Instructions

Table 3-5 gives the protocols followed for each Floating-Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Appendix A.

The Operand class columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see *Series 32000 Instruction Set Reference Manual*).

The Operand Issued columns show the sizes of the operands issued to the Floating-Point Unit by the CPU. "D" indicates a 32-bit Double Word, "I" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). "f" indicates that the instruction specifies a Floating-Point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the Slave Processor Status Word (*Figure 3-26*).

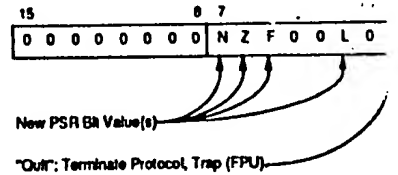


FIGURE 3-26. Slave Processor Status Word Format

Any operand indicated as being of type "f" will cause a transfer if the Register addressing mode specified. This is because the Floating-Point Regist are physically on the Floating-Point Unit and therefore available without CPU assistance.

TABLE 3-5. Floating-Point Instruction Protocols

Mnemonic	Operand 1 Class	Operand 2 Class	Operand 1 Issued	Operand 2 Issue	Returned Value Type and Dest	PSR Bits Affected
ADDf	read.f	rmw.f	f	f	f to Op.2	none
SUBf	read.f	rmw.f	f	f	f to Op.2	none
MULf	read.f	rmw.f	f	f	f to Op.2	none
DIVf	read.f	rmw.f	f	f	f to Op.2	none
MOVf	read.f	write.f	f	N/A	f to Op.2	none
ABSf	read.f	write.f	f	N/A	f to Op.2	none
NEGf	read.f	write.f	f	N/A	f to Op.2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op.2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op.2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op.2	none
MOVFL	read.F	write.L	F	N/A	L to Op.2	none
MOVLF	read.L	write.F	L	N/A	F to Op.2	none
MOVif	read.i	write.f	i	N/A	f to Op.2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op.2	none
POLYf	read.f	read.f	f	f	f to F0	none
DOTf	read.f	read.f	f	f	f to F0	none
SCALBf	read.f	rmw.f	f	f	f to Op.2	none
LOGBf	read.f	write.f	f	N/A	f to Op.2	none

Note:
D = Double Word
i = Integer size (B, W, D) specified in mnemonic.
f = Floating-Point type (F, L) specified in mnemonic.
N/A = Not Applicable to this instruction.

5) Set "Return Address" to the address of the first byte of the trapped instruction.

6) Perform Service (Vector, Return Address), *Figure 3-24.*

3.7.6.3 Trace Trap Sequence

1) In the Processor Status Register (PSR), clear the P bit.

2) Copy the PSR into a temporary register, then clear PSR bits S, U and T.

3) Push the PSR copy onto the Interrupt Stack as a 16-bit value.

4) Set "Vector" to 9.

5) Set "Return Address" to the address of the next instruction.

6) Perform Service (Vector, Return Address), *Figure 3-24.*

3.8 SLAVE PROCESSOR INSTRUCTIONS

The NS32FX16 supports only one group of instructions, the floating-point instruction set, as being executable by a slave processor. The floating-point instruction set is validated by the F bit in the CFG register.

If a floating-point instruction is encountered and the F bit in the CFG register is not set, a Trap (UND) will result, without any slave processor communication attempted by the CPU. This allows software emulation in case an external floating-point unit (FPU) is not used.

3.8.1 Slave Processor Protocol

Slave Processor instructions have a three-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word. The ID Byte has three functions:

1) It identifies the instruction as being a Slave Processor Instruction.
2) It specifies which Slave Processor will execute it.
3) It determines the format of the following Operation Word of the instruction.

Upon receiving a Slave Processor Instruction, the CPU initiates the sequence outlined in *Figure 3-25.* While applying Status Code 1111 (Broadcast ID, Section 3.4.1), the CPU transfers the ID Byte on the least-significant half of the Data Bus (AD0-AD7). All Slave Processors input this byte and decode it. The Slave Processor selected by the ID Byte is activated, and from this point the CPU is communicating only with it. If any other slave protocol was in progress (e.g., an aborted Slave instruction), this transfer cancels it.

Status Combinations:
Send ID (ID): Code 1111
Xfer Operand (OP): Code 1101
Read Status (ST): Code 1110

Step	Status	Action
1	ID	CPU Sends ID Byte.
2	OP	CPU Sends Operation Word.
3	OP	CPU Sends Required Operands.
4	—	Slave Starts Execution. CPU Prefetches.
5	—	Slave Pulses $\overline{SPC}$ Low.
6	ST	CPU Reads Status Word. (Trap? Alter Flags?)
7	OP	CPU Reads Results (If Any).

FIGURE 3-25. Slave Processor Protocol

The CPU next sends the Operation Word while applying Status Code 1101 (Transfer Slave Operand, Section 3.4.1). Upon receiving it, the Slave Processor decodes it, and at this point both the CPU and the Slave Processor are aware of the number of operands to be transferred and their sizes. The Operation Word is swapped on the Data Bus; that is, bits 0-7 appear on pins AD8-AD15 and 8-15 appear on pins AD0-AD7.

Using the Addressing Mode fields within the Operation Word, the CPU starts fetching operands and issuing them to the Slave Processor. To do so, it references any Addressing Mode extensions which may be appended to the Slave Processor instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the Slave Processor. The Status Code applied is 1101 (Transfer Slave Processor Operand, Section 3.4.1).

After the CPU has issued the last operand, the Slave Processor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low.

While the Slave Processor is executing the instruction, the CPU is free to prefetch instructions into its queue. If it fills the queue before the Slave Processor finishes, the CPU will wait, applying Status Code 0011 (Waiting for Slave).

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a Status Word from the Slave Processor, applying Status Code 1110 (Read Slave Status). This word has the format shown in *Figure 3-26.* If the Q bit ("Quit", Bit 0) is set, this indicates that an error was detected by the Slave Processor. The CPU will not continue the protocol, but will immediately trap through the Slave vector in the Interrupt Table. Certain Slave Processor instructions cause CPU PSR bits to be loaded from the Status Word.

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The Read cycles from the Slave Processor are performed by the CPU while applying Status Code 1101 (Transfer Slave Operand).

3.4.2 Basic Read and Write Cycles

The sequence of events occurring during a CPU access to either memory or peripheral device is shown in *Figure 3-7* for a read cycle, and *Figure 3-8* for a write cycle.

The cases shown assume that the selected memory or peripheral device is capable of communicating with the CPU at full speed. If not, then cycle extension may be requested through CWAIT and/or WAIT1-2.

A full-speed bus cycle is performed in four cycles of the CTTL clock signal, labeled T1 through T4. Clock cycles not associated with a bus cycle are designated Ti (for "idle").

During T1, the CPU applies an address on pins AD0-AD15 and A16-A23. It also provides a low-going pulse on the $\overline{ADS}$ pin, which serves the dual purpose of informing external circuitry that a bus cycle is starting and of providing control to an external latch for demultiplexing Address bits 0-15 from the AD0-AD15 pins. See *Figure 3-6.*

However, using the ALE output signal is suggested for controlling the address latch. In normal CPU read cycles, and in external DMA cycles, ALE is asserted (high) at T4, and is deasserted (low) at T1 (see Figure 3.7). This eliminates the need for inverting the existing $\overline{ADS}$ off-chip to generate the address latch strobe, and removes the address latch strobe from the critical path to memory.

In CPU read and write cycles that access the on-chip FAM, in slave cycles and in non-DMA idle states, ALE is always high. ALE is active (high) after reset. ALE is never Tri-State.

During this time also the status signals $\overline{DDIN}$, indicating the direction of the transfer, and $\overline{HBE}$, indicating whether the high byte (AD8-AD15) is to be referenced, become valid.

During T2 the CPU switches the Data Bus, AD0-AD15, to either accept or present data. Note that the signals A16-A23 remain valid and need not be latched.

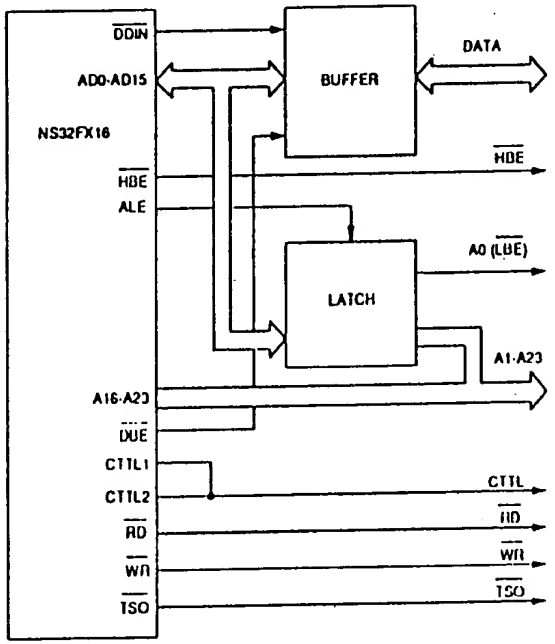


FIGURE 3-6. Bus Connections

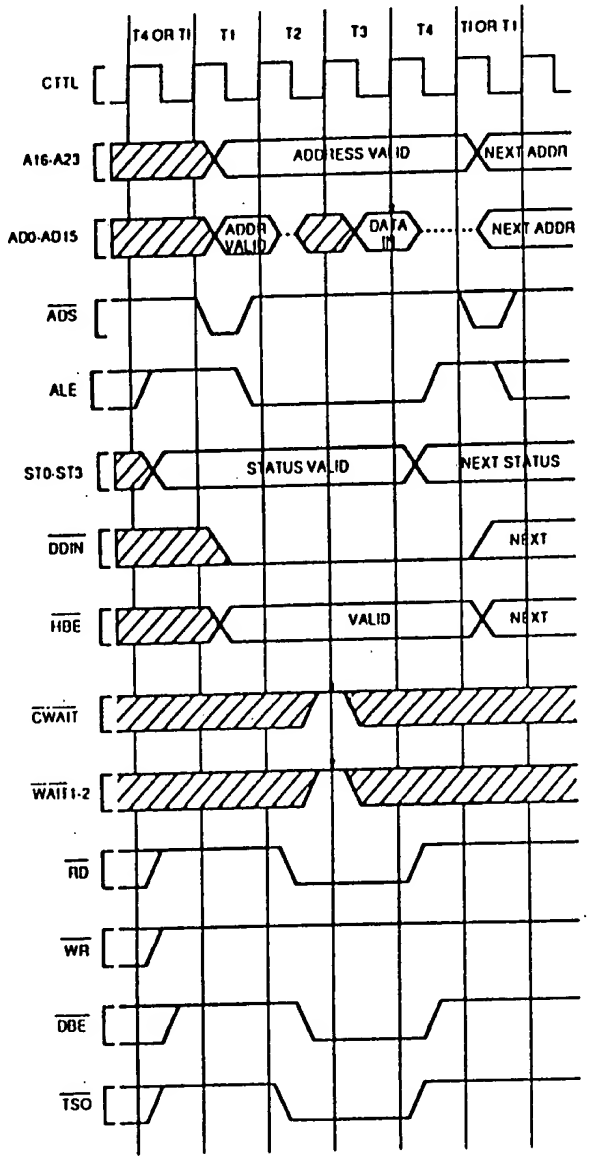


FIGURE 3-7. Off-Chip Read Cycle Timing

3.7.7 Priority Among Exceptions

The NS32FX16 CPU internally prioritizes simultaneous interrupt and trap requests as follows:

1) Traps other than Trace (Highest priority)
2) Non-Maskable Interrupt
3) Maskable Interrupts
4) Trace Trap (Lowest priority)

3.7.8 Exception Acknowledge Sequences: Detail Flow

For purposes of the following detailed discussion of interrupt and trap acknowledge sequences, a single sequence called "Service" is defined in *Figure 3-24*. Upon detecting any interrupt request or trap condition, the CPU first performs a sequence dependent upon the type of interrupt or trap. This sequence will include pushing the Processor Status Register and establishing a Vector and a Return Address. The CPU then performs the Service sequence.

3.7.8.1 Maskable/Non-Maskable Interrupt Sequence

This sequence is performed by the CPU when the NMI pin receives a falling edge, or the INT pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or at the next interruptible point during its execution, as in the case of string or graphic instructions that have interior loops. The graphics instructions are interruptible.

1. If a String Instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P bit.
   b. Set "Return Address" to the address of the first byte of the interrupted instruction. Otherwise, set "Return Address" to the address of the next instruction.
2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, T, P and I.
3. If the interrupt is Non-Maskable:
   a. Read a byte from address $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowlege, Master: Section 3.4.1). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.
4. If the interrupt is Non-Vectored:
   a. Read a byte from address $FFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Section 3.4.1). Discard the byte read.
   b. Set "Vector" to 0.
   c. Go to Step 8.
5. Here the Interrupt is Vectored. Read "Byte" from address $FFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master: Section 3.4.1).
6. If "Byte" >= 0, then set "Vector" to "Byte" and go to Step 8.
7. If "Byte" is in the range -16 through -1, then the interrupt source is Cascaded. (More negative values are reserved for future use.) Perform the following:
   a. Read the 32-bit Cascade Address from memory. The address is calculated as INTBASE + 4 * Byte.
   b. Read "Vector" applying the Cascade Address just read and Status Code 0101 (Interrupt Acknowledge, Cascaded: Section 3.4.1).
8. Push the PSR copy (from Step 2) onto the Interrupt Stack as a 16-bit value.
9. Perform Service (Vector, Return Address). *Figure 3-24.*

Service (Vector, Return Address):

1) Read the 32-bit External Procedure Descriptor for the Interrupt Dispatch Table: address is Vector*4+INTBASE Register contents.
2) Move the Module field of the Descriptor into the temporary MOD Register.
3) Read the Program Base pointer from memory address MOD + 8, and add it to the Offset field from the Descriptor, placing the result in the Program Counter.
4) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.
5) Flush Queue: Non-sequentially fetch first instruction of Interrupt Routine.
6) Push MOD Register onto the Interrupt Stack as a 16-bit value. (The PSR has already been pushed as a 16-bit value.)
7) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.
8) Copy temporary MOD Register to MOD Register.

FIGURE 3-24: Service Sequence Invoked during All Interrupt/Trap Sequences

3.7.8.2 Trap Sequence: Traps Other Than Trace

1) Restore the currently selected Stack Pointer and the Processor Status Register to their original values at the start of the trapped instruction.
2) Set "Vector" to the value corresponding to the trap type.

SLAVE:	Vector=3
ILL:	Vector=4.
SVC:	Vector=5.
DVZ:	Vector=6.
FLG:	Vector=7.
BPT:	Vector=8.
UND:	Vector=10.

3) Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, P and T.
4) Push the PSR copy onto the Interrupt Stack as a 16 bit value.

3.7.4 Non-Maskable Interrupt

The Non-Maskable Interrupt is triggered whenever a falling edge is detected on the NMI pin. The CPU performs an "Interrupt Acknowledge, Master" bus cycle when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is $FFFF00_{16}$. The vector value used for the Non-Maskable Interrupt is taken as 1, regardless of the value read from the bus.

The service procedure returns from the Non Maskable Interrupt using the Return from Trap (RETT) instruction. No special bus cycles occur on return.

For a full sequence of events in processing the Non-Maskable Interrupt, see Section 3.7.8.1.

3.7.5 Traps

Traps are processing exceptions that are generated as direct results of the execution of an instruction. The Return Address pushed by any trap except Trap (TRC) is the address of the first byte of the instruction during which the trap occurred. Traps do not disable interrupts, as they are not associated with external events. Traps recognized by NS32FX16 CPU are:

Trap (SLAVE): An exceptional condition was detected by the Floating Point Unit during the execution of a Slave Instruction. This trap is requested via the Status Word returned as part of the Slave Processor Protocol (Section 3.8.1).

Trap (ILL): Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR bit U=1).

Trap (SVC): The Supervisor Call (SVC) instruction was executed.

Trap (DVZ): An attempt was made to divide an integer by zero. (The SLAVE trap is used for Floating Point division by zero.)

Trap (FLG): The FLAG instruction detected a "1" in the PSR F bit.

Trap (BPT): The Breakpoint (BPT) instruction was executed.

Trap (TRC): The Instruction just completed is being traced. See Section 3.7.6.

Trap (UND): An undefined opcode was encountered by the CPU.

3.7.6 Instruction Tracing

Instruction tracing is a feature that can be used during debugging to single-step through selected portions of a program. Tracing is enabled by setting the T-bit in the PSR Register. When enabled, the CPU generates a Trace Trap (TRC) after the execution of each instruction.

At the beginning of each instruction, the T bit is copied into the PSR P (Trace "Pending") bit. If the P bit is set at the end of an instruction, then the Trace Trap is activated. If any other trap or interrupt request is made during a traced instruction, its entire service procedure is allowed to complete before the Trace Trap occurs. Each interrupt and trap sequence handles the P bit for proper tracing, guaranteeing only one Trace Trap per instruction, and guaranteeing that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

Due to the fact that some instructions can clear the T and P bits in the PSR, in some cases a Trace Trap may not occur at the end of the instruction. This happens when one of the privileged instructions, BICPSRW or LPRW PSR, is executed.

In other cases, it is still possible to guarantee that a Trace Trap occurs at the end of the instruction, provided that special care is taken before returning from the Trace Trap Service Procedure. In case a BICPSRB instruction has been executed, the service procedure should make sure that the T bit in the PSR copy saved on the Interrupt Stack is set before executing the RETT instruction to return to the program being traced. If the RETT or RETI instructions have to be traced, the Trace Trap Service Procedure should set the P and T bits in the PSR copy of the Interrupt Stack that is going to be restored in the execution of such instructions.

While debugging the NS32FX16 instructions which have interior loops (BBOR, BBXOR, BBAND, BBFOR, EXTBLT, MOVMP, SBITPS, TBITS), special care must be taken with the single-step trap. If an interrupt occurs during a single-step of one of the graphics instructions, the interrupt will be serviced. Upon return from the interrupt service routine, the new NS32FX16 instruction will not be re-entered, due to a single-step trap. Both the $\overline{NMI}$ and $\overline{INT}$ interrupts will cause this behavior. Another single-step operation (S command in DBUG/MON16) will resume from where the instruction was interrupted. There are no side effects from this early termination, and the instruction will complete normally.

For all other Series 32000 instructions, a single-step operation will complete the entire instruction before trapping back to the debugger. On the instructions mentioned above, several single-step commands may be required to complete the instruction, ONLY when interrupts are occurring.

There are some suggested methods to give the appearance of single-stepping for these NS32FX16 instructions.

1. MON16 monitors the return from the single-step trap vector's PC value. If the PC has not changed since the last single-step command was issued, the single-step operation is repeated. It is also advisable to ensure that one of the NS32FX16 instructions is being single-stepped by inspecting the first byte of the address pointed to by the PC register. If it is 0x0E, then the instruction is an NS32FX16 specific instruction.
2. A breakpoint following the instruction would also trap after the instruction had completed.

Note: If instruction tracing is enabled while the wait instruction is executed, the Trap (TRC) occurs after the next interrupt, when the interrupt service procedure has returned

3.0 Functional Description (Continued)

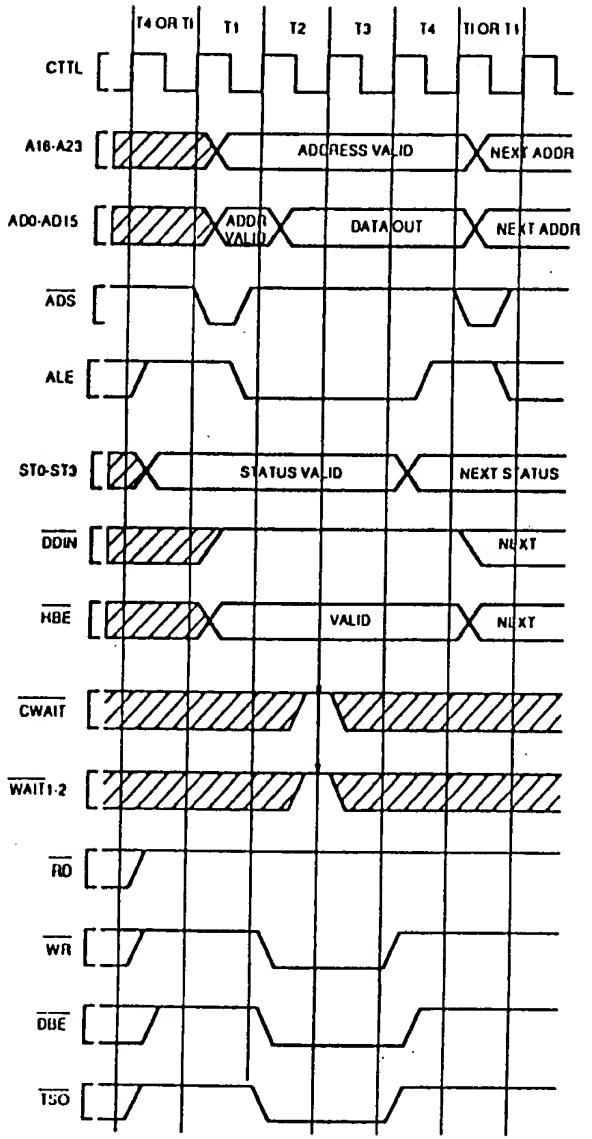


FIGURE 3-8. Off-Chip Write Cycle Timing

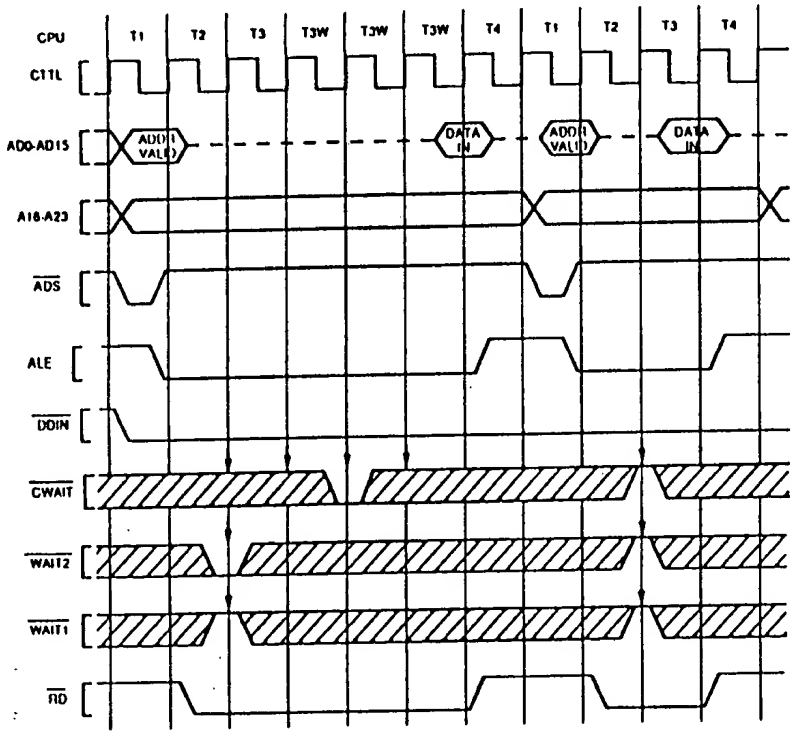


FIGURE 3-9. Extension of an Off-Chip Read Cycle

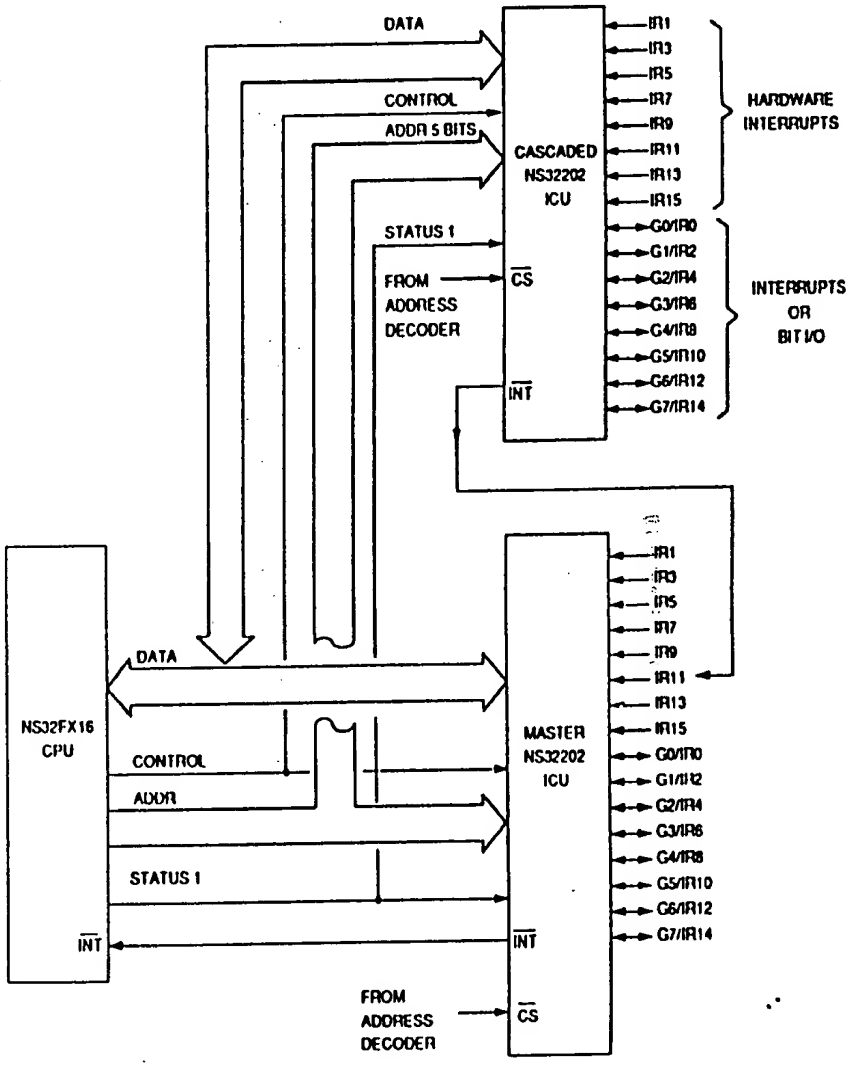


FIGURE 3-23. Cascaded Interrupt Control Unit Connections

3.0 Functional Description (Continued)

In a system which uses cascading, two tasks must be performed upon initialization:

1) For each Cascaded ICU in the system, the Mater ICU must be informed of the line number (0 to 15) on which it receives the cascaded requests.

2) A Cascade Table must be established in memory. The Cascade Table is located in a NEGATIVE direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

Figure 3-18 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0 to 15) and subtract 16 from it, giving an index in the range -16 to -1. Multiply this value by 4, and add the resulting negative number to the contents of the INTBASE Register. The 32-bit entry at this address must be set to the address of the Hardware Vector Register of the Cascaded ICU. This is referred to as the "Cascade Address."

Upon receipt of an interrupt request from a Cascaded ICU, the Master ICU interrupts the CPU and provides the negative Cascade Table index instead of a (positive) vector number. The CPU, seeing the negative value, uses it as an index into the Cascade Table and reads the Cascade Address from the referenced entry. Applying this address, the CPU performs an "Interrupt Acknowledge, Cascaded" bus cycle, reading the final vector value. This vector is interpreted by the CPU as an unsigned byte and can therefore be in the range 0 through 255.

In returning from a Cascaded interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any Maskable Interrupt. The CPU performs an "End of Interrupt, Master" bus cycle, whereupon the Master ICU again provides the negative Cascaded Table index. The CPU, seeing a negative value, uses it to find the corresponding Cascade Address from the Cascade Table. Applying this address, it performs an "End of Interrupt, Cascaded" bus cycle, informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

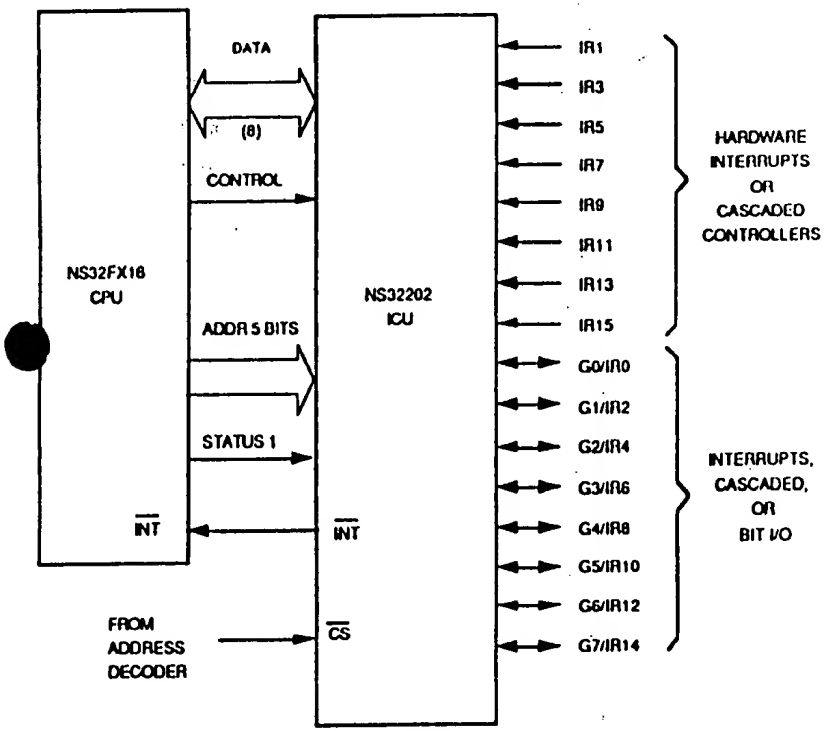


FIGURE 3-22. Interrupt Control Unit Connections (16 Levels)

Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the Interrupt Mask Register of the Interrupt Controller. However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the INT line before the ICU deasserted it. This could cause the ICU to provide an invalid vector. To avoid this problem the above operation should be performed with the CPU interrupt disabled.

3.0 Functional Description (Continued)

At this time the signals $\overline{TSO}$ (Timing State Output), $\overline{DBE}$ (Data Buffer Enable) and either $\overline{RD}$ (Read Strobe) or $\overline{WR}$ (Write Strobe) will also be activated.

The T3 state provides for access time requirements, and it occurs at least once in a bus cycle. At the end of T2, on the rising edge of CTTL, the $\overline{CWAIT}$ and $\overline{WAIT}$1-2 signals are sampled to determine whether the bus cycle will be extended. See Section 3.4.3.

If the CPU is performing a read cycle, the data bus (AD0-AD15) is sampled at the beginning of T4 on the rising edge of CTTL. Data must, however, be held a little longer to meet the data hold time requirements. The $\overline{RD}$ signal is guaranteed not to go inactive before this time, so its rising edge can be safely used to disable the device providing the input data.

The T4 state finishes the bus cycle. At the beginning of T4, the $\overline{RD}$ or $\overline{WR}$, and $\overline{TSO}$ signals go inactive, and on the falling edge of CTTL, $\overline{DBE}$ goes inactive, having provided for necessary data hold times. Data during Write cycles remains valid from the CPU throughout T4. Note that the Bus Status lines (ST0-ST3) change at the beginning of T4, anticipating the following bus cycle (if any).

3.4.3 Cycle Extension

To allow sufficient access time for any speed of memory or peripheral device, the NS32FX16 provides for extension of a bus cycle. Any type of bus cycle except a Slave Processor cycle can be extended.

In *Figures 3-7* and *3-8*, note that during T3 all bus control signals from the CPU are flat. Therefore, a bus cycle can be clearly extended by causing the T3 state to be repeated. This is the purpose of the $\overline{WAIT}$1-2 and $\overline{CWAIT}$ input signals.

At the end of state T2, on the rising edge of CTTL, $\overline{WAIT}$1-2 and $\overline{CWAIT}$ are sampled.

If any of these signals are active, the bus cycle will be extended by at least one clock cycle. Thus, one or more additional T3 states (also called wait state T3W) will be inserted after the next T-State. Any combination of the above signals can be activated at one time. However, the $\overline{WAIT}$1-2 inputs are only sampled by the CPU at the end of state T2. They are ignored at all other times.

The $\overline{WAIT}$1-2 inputs are binary weighted, and can be used to insert up to 3 wait states, according to the following table.

$\overline{WAIT}$2	$\overline{WAIT}$1	Number of Wait States
HIGH	HIGH	0
HIGH	LOW	1
LOW	HIGH	2
LOW	LOW	3

$\overline{CWAIT}$ causes wait states to be inserted continuously as long as it is sampled active. It is normally used when the number of wait states to be inserted in the CPU bus cycle is not known in advance.

The following sequence shows the CPU response to the $\overline{WAIT}$1-2 and $\overline{CWAIT}$ inputs.

1. Start bus cycle.
2. Sample $\overline{WAIT}$1-2 and $\overline{CWAIT}$ at the end of state T2.
3. If the $\overline{WAIT}$1-2 inputs are both inactive, then go to step 6.
4. Insert the number of wait states selected by $\overline{WAIT}$1-2.
5. Sample $\overline{CWAIT}$ again.
6. If $\overline{CWAIT}$ is not active, then go to step 8.
7. Insert one wait state and then go to step 5.
8. Complete bus cycle.

Figure 3-9 shows a bus cycle extended by three wait states, two of which are due to $\overline{WAIT}$2 and one of which is due to $\overline{CWAIT}$.

3.4.4 Data Access Sequences

The 24-bit address provided by the NS32FX16 is a byte address; that is, it uniquely identifies one of up to 16,777,216 eight-bit memory locations. An important feature of the NS32FX16 is that the presence of a 16-bit data bus imposes no restrictions on data alignment; any data item, regardless of size, may be placed starting at any memory address. The NS32FX16 provides a special control signal, High Byte Enable ($\overline{HBE}$), which facilitates individual byte addressing on a 16-bit bus.

Memory is organized as two eight-bit banks, each bank receiving the word address (A1-A23) in parallel. One bank, connected to Data Bus pins AD0-AD7, is enabled to respond to even byte addresses; i.e., when the least significant address bit (A0) is low. The other bank, connected to Data Bus pins AD8-AD15, is enabled when $\overline{HBE}$ is low. See *Figure 3-10*.

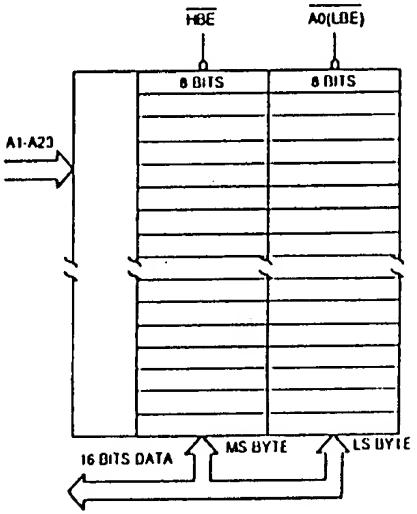


FIGURE 3-10. Memory Interface

Any bus cycle falls into one of three categories: Even Byte Access, Odd Byte Access, and Even Word Access. All accesses to any data type are made up of sequences of these cycles. Table 3-2 gives the state of A0 and $\overline{HBE}$ for each category.

TABLE 3-2. Bus Cycle Categories

Category	$\overline{HBE}$	A0
Even Byte	1	0
Odd Byte	0	1
Even Word	0	0

Accesses of operands requiring more than one bus cycle are performed sequentially, with no idle T-States separating them. The number of bus cycles required to transfer an operand depends on its size and its alignment (i.e., whether it starts on an even byte address or an odd byte address). Table 3-3 lists the bus cycle performed for each situation. For the timing of A0 and $\overline{HBE}$, see Section 3.4.2.

3.4.4.1 Bit Accesses

The Bit Instructions perform byte accesses to the byte containing the designated bit. The Test and Set Bit instruction (SBIT), for example, reads a byte, alters it, and rewrites it, having changed the contents of one bit.

3.4.4.2 Bit Field Accesses

An access to a Bit Field in memory always generates a Double-Word transfer at the address containing the least significant bit of the field. The Double Word is read by an Extract Instruction; an Insert instruction reads a Double Word, modifies it, and rewrites it.

3.4.4.3 Extended Multiple Accesses

The Multiply Extended Integer (MEI) instruction will return a result which is twice the size in bytes of the operand it reads. If the multiplicand is in memory, the most-significant half of the result is written first (at the higher address), then the least-significant half.

3.4.5 Instruction Fetches

Instructions for the NS32FX16 CPU are "prefetched"; that is, they are input before being needed into the next available entry of the eight-byte Instruction Queue. The CPU performs two types of Instruction Fetch cycles, Sequential and Non-Sequential. These can be distinguished from each other by their differing status combinations on pins ST0-ST3 (Section 3.4.1).

A Sequential Fetch will be performed by the CPU whenever the Data Bus would otherwise be idle and the Instruction Queue is not currently full. Sequential Fetches are always Even Word Read cycles (Table 3-2). A Non-Sequential Fetch occurs as a result of any break in the normally sequential flow of a program. Any jump or branch instruction, a trap or an interrupt, will cause the next Instruction Fetch cycle to be Non-Sequential.

In addition, certain instructions flush the instruction queue, causing the next instruction fetch to display Non-Sequential status. Only the first bus cycle after a break displays Non-Sequential status, and that cycle is either an Even Word Read or an Odd Byte Read, depending on whether the destination address is even or odd.

3.4.6 Interrupt Control Cycles

Activating the $\overline{INT}$ or $\overline{NMI}$ pin on the CPU will initiate one or more bus cycles whose purpose is interrupt control rather than the transfer of instructions or data. Execution of the Return from Interrupt instruction (RETI) will also cause Interrupt Control bus cycles. These differ from instruction or data transfers only in the status presented on pins ST0-ST3. All Interrupt Control cycles are single-byte Read cycles.

Table 3-4 shows the Interrupt Control sequences associated with each interrupt and with the return from its service routine. For full details of the NS32FX16 interrupt structure, see Section 3.7.

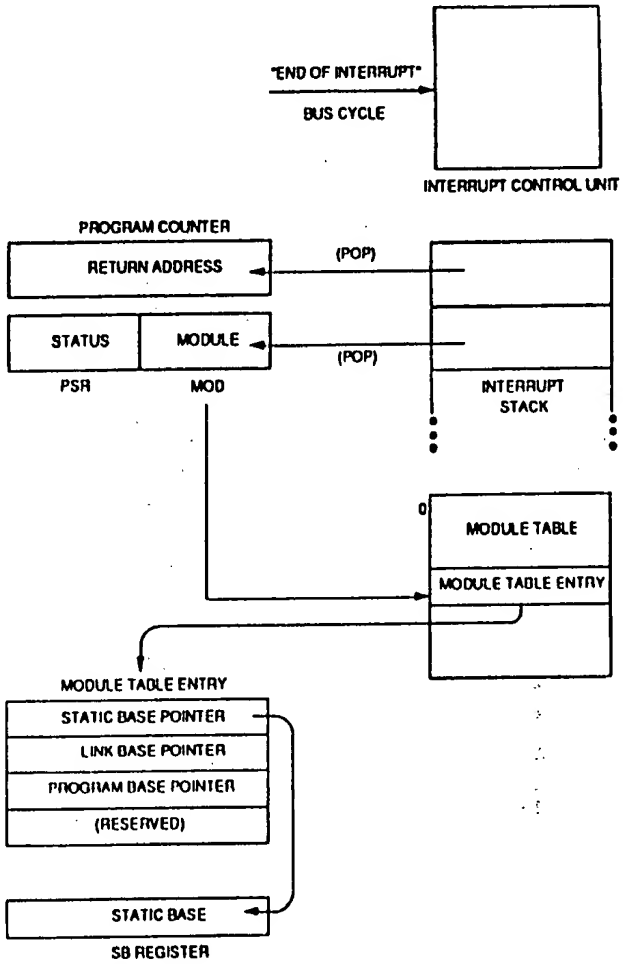


FIGURE 3-21. Return from Interrupt (RETI) Instruction Flow

3.7.3.2 Vectored Mode: Non-Cascaded Case

In the Vectored mode, the CPU uses an Interrupt Control Unit (ICU) to prioritize up to 16 interrupt requests. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an "Interrupt Acknowledge Master" bus cycle reading a vector value from the low-order byte of the Data Bus. This vector is then used as an index into the Dispatch Table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) Instruction, which performs an End of Interrupt bus cycle, informing the ICU that it may re-prioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it needs also to inform a Cascaded ICU.

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range 0 through 127; that is, they must be positive numbers in eight bits. By providing a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU (see below).

3.7.3.3 Vectored Mode: Cascaded Case

In order to allow up to 256 levels of interrupt, provision is made both in the CPU and in the NS32202 Interrupt Control Unit (ICU) to transparently support cascading. *Figure 3-23* shows a typical cascaded configuration. Note that the Interrupt output from a Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin.

3.0 Functional Description (Continued)

3.7.2 Returning from an Exception Service Procedure

To return control to an interrupted program, one of two instructions can be used: RETT (Return from Trap) and RETI (Return from Interrupt).

RETT is used to return from any trap or a non-maskable interrupt service procedure. Since some traps are often used deliberately as a call mechanism for supervisor mode procedures, RETT can also adjust the Stack Pointer (SP) to discard a specified number of bytes from the original stack as surplus parameter space.

RETI is used to return from a maskable interrupt service procedure. A difference of RETT, RETI also informs any external interrupt control units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not discard parameters from the stack.

Both of the above instructions always restore the PSR, MOD, PC and SB registers to their previous contents.

3.7.3 Maskable Interrupts

The $\overline{INT}$ pin is a level-sensitive input. A continuous low level is allowed for generating multiple interrupt requests. The input is maskable, and is therefore enabled to generate interrupt requests only while the Processor Status Register I bit is set. The I bit is automatically cleared during service of an $\overline{INT}$ or NMI request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

The $\overline{INT}$ pin may be configured via the SETCFG instruction as either Non-Vectored (CFG Register bit I=0) or Vectored (bit I=1).

3.7.3.1 Non-Vectored Mode

In the Non-Vectored mode, an interrupt request on the INT pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.

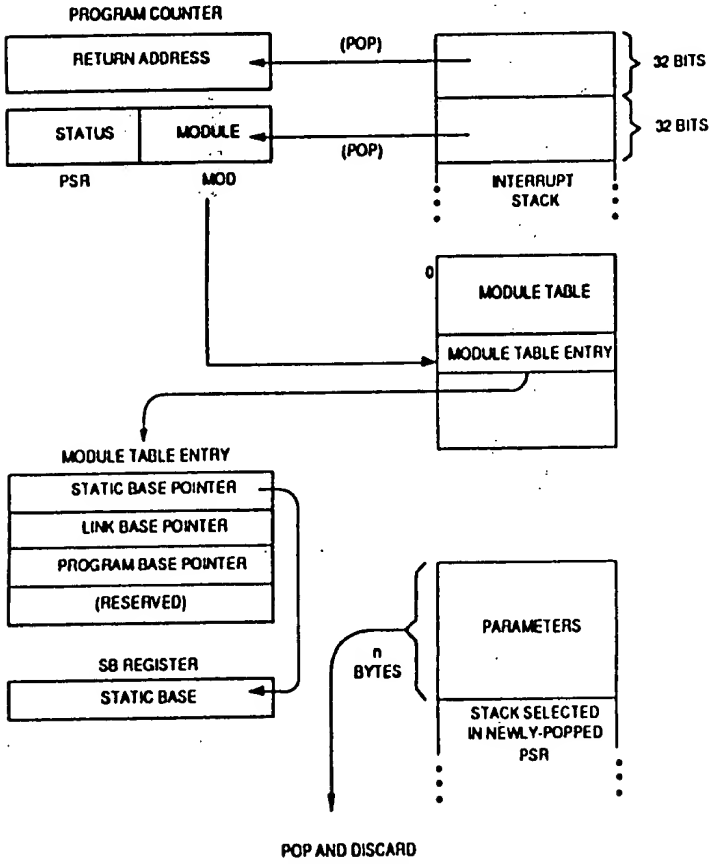


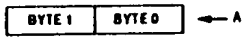
FIGURE 3-20. Return from Trap (RETT) Instruction Flow

3.0 Functional Description (Continued)

TABLE 3-3. Access Sequences

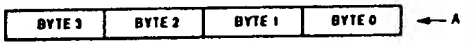
Cycle	Type	Address	$\overline{HBE}$	A0	High Bus	Low Bus

A. Odd Word Access Sequence



1	Odd Byte	A	0	1	Byte 0	Don't Care
2	Even Byte	A + 1	1	0	Don't Care	Byte 1

B. Even Double-Word Access Sequence



1	Even Word	A	0	0	Byte 1	Byte 0
2	Even Word	A + 2	0	0	Byte 3	Byte 2

C. Odd Double-Word Access Sequence

1	Odd Byte	A	0	1	Byte 0	Don't Care
2	Even Word	A + 1	0	0	Byte 2	Byte 1
3	Even Byte	A + 3	1	0	Don't Care	Byte 3

D. Even Quad-Word Access Sequence

1	Even Word	A	0	0	Byte 1	Byte 0
2	Even Word	A + 2	0	0	Byte 3	Byte 2
Other bus cycles (instruction prefetch or slave) can occur here.						
3	Even Word	A + 4	0	0	Byte 5	Byte 4
4	Even Word	A + 6	0	0	Byte 7	Byte 6

E. Odd Quad-Word Access Sequence

1	Odd Byte	A	0	1	Byte 0	Don't Care
2	Even Word	A + 1	0	0	Byte 2	Byte 1
3	Even Byte	A + 3	1	0	Don't Care	Byte 3
Other bus cycles (instruction prefetch or slave) can occur here.						
4	Odd Byte	A + 4	0	1	Byte 4	Don't Care
5	Even Word	A + 5	0	0	Byte 6	Byte 5
6	Even Byte	A + 7	1	0	Don't Care	Byte 7

TABLE 3-4. Interrupt Sequences

Cycle	Status	Address	$\overline{DDIN}$	$\overline{HBE}$	A0	High Bus	Low Bus
A. Non-Maskable Interrupt Control Sequence							
Interrupt Acknowledge							
1	0100	$FFFF00_{16}$	0	1	0	Don't Care	Don't Care
Interrupt Return							
None: Performed through Return from Trap (RETT) Instruction.							
B. Non-Vectored Interrupt Control Sequence							
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Don't Care
Interrupt Return							
None: Performed through Return from Trap (RETT) Instruction.							
C. Vectored Interrupt Sequence: Non-Cascaded							
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Vector: Range: 0-127
Interrupt Return							
1	0110	$FFFE00_{16}$	0	1	0	Don't Care	Vector: Same as in Previous Int Ack. Cycle
D. Vectored Interrupt Sequence: Cascaded							
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Cascade Index: range - 16 to -1
(The CPU here uses the Cascade Index to find the Cascade Address.)							
2	0101	Cascade Address	0 0*	1 or 1*	0 or	Vector, range 0-255; on appropriate half of Data Bus for even/odd address	
Interrupt Return							
1	0110	$FFFE00_{16}$	0	1	0	Don't Care	Cascade Index: same as in previous Int. Ack. Cycle
(The CPU here uses the Cascade Index to find the Cascade Address.)							
2	0111	Cascade Address	0	1 or 0*	0 or 1*	Don't Care	Don't Care

* If the Cascaded ICU address is Even (A0 is low), then the CPU applies $\overline{HBE}$ high and reads the vector number from bits 0-7 of the Data Bus.

If the address is Odd (A0 is high), then the CPU applies $\overline{HBE}$ low and reads the vector number from bits 8-15 of the Data Bus. The vector number may be in the range 0-255.

This process is illustrated in *Figure 3-19* from the view point of the programmer.

Details on the sequences of events in processing interrupts and traps are given in the following sections.

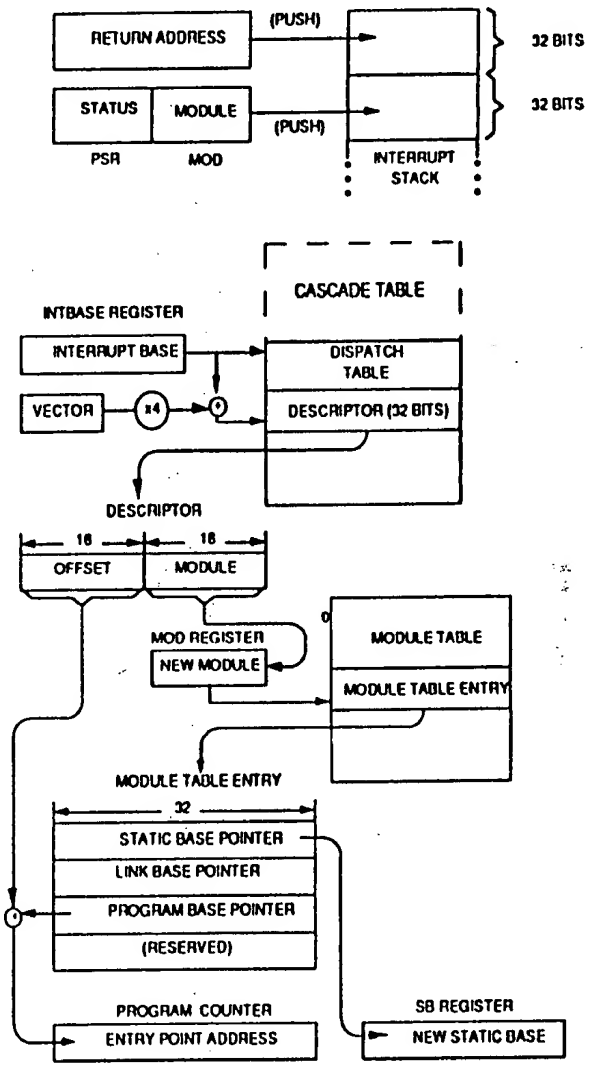


FIGURE 3-19. Exception Acknowledge Sequence

3.7 EXCEPTION PROCESSING

Exceptions are special events that alter the sequence of instruction execution. The CPU recognizes two basic types of exceptions: interrupts and traps.

An interrupt occurs in response to an event signalled by activating the NMI or INT input signals. Interrupts are typically requested by peripheral devices that require the CPU's attention.

Traps occur as a result either of exceptional conditions (e.g. attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g. supervisor call instruction).

When an exception is recognized, the CPU saves the PC, PSR and the MOD register contents on the interrupt stack and then it transfers control to an exception service procedure.

Details on the operations performed in the various cases by the CPU to enter and exit the exception service procedure are given in the following sections.

It is to be noted that the reset operation is not treated here as an exception, even though, like any exception, it alters the instruction execution sequence. This is because the CPU handles reset in a significantly different way than it handles exceptions.

Refer to Section 3.3 for details on the reset operation.

3.7.1 Exception Acknowledge Sequence

When an exception is recognized, the CPU goes through three major steps:

1) Adjustment of Registers.

Depending on the source of the exception, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR) and the currently-selected Stack Pointer (SP). A copy of the PSR is made and the PSR is then set to reflect Supervisor Mode and selection of the Interrupt Stack.

2) Vector Acquisition

A Vector is either obtained from the Data Bus or is supplied by default.

3) Service Call

The Vector is used as an index into the Interrupt Dispatch Table, whose base address is taken from the CPU Interrupt Base (INTBASE) Register. See *Figure 3-18*. A 32-bit External Procedure Call is performed using it. The MOD Register (16 bits) and Program Counter (32 bits) are pushed on the Interrupt Stack.

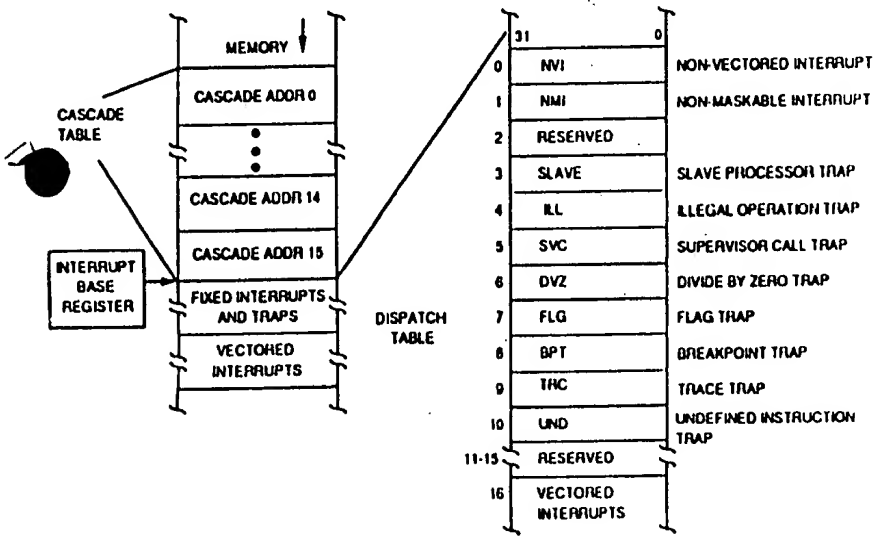


FIGURE 3-18. Interrupt Dispatch and Cascade Tables

3.4.7 On-Chip Bus Cycles

The bus cycles accessing registers of the on-chip FAX Accelerator Module do not involve any off-chip resource. However, for observability reasons, the NS32FX16's bus interface provides all the necessary information in order to allow a debug or trace device (e.g. ISE) to track an on-chip bus transaction.

An on-chip bus transaction is very similar (timewise) to an off-chip bus transaction. However, the ADS, RD, WR, TSO, and DBE outputs are not asserted by the CPU. Instead, the NS32FX16 asserts a special output, IAS. During write cycles to on-chip addresses, the data to be written can be observed on AD0-AD15.

Access to the FAM registers while it is executing a vector operation are delayed (as if the CWAIT input is active). When the FAM finishes the operation, access to the registers proceeds. These wait states cannot be observed on external pins.

The address on AD0-AD15 and A16-A23 during internal reference is the 24 least significant bits of the addressed internal register address.

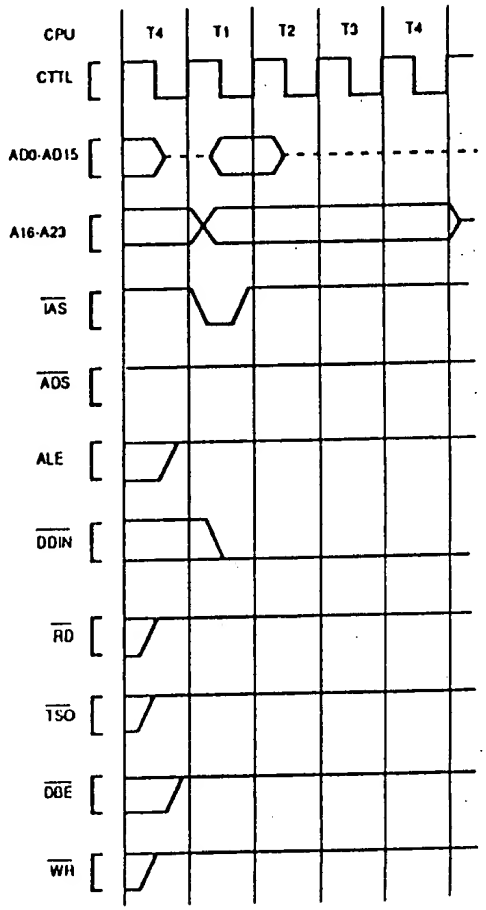


FIGURE 3-11 (a). On-Chip Read Cycle

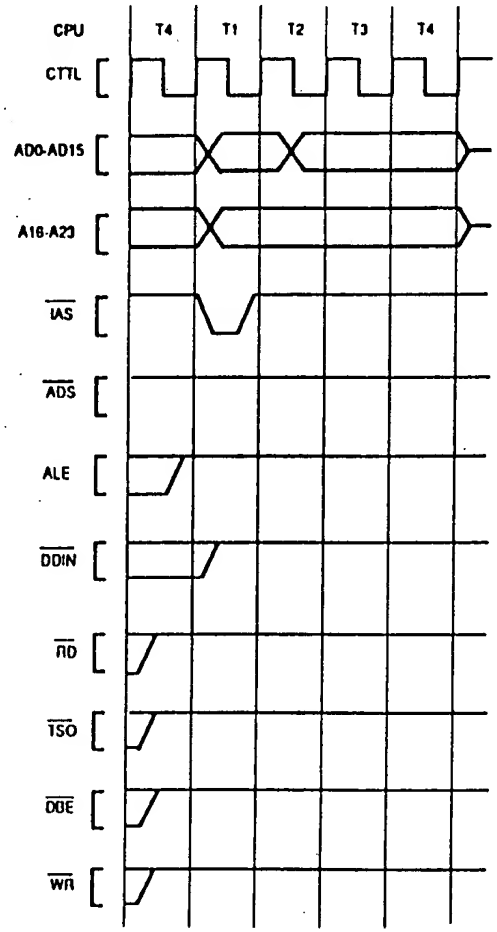


FIGURE 3-11 (b). On-Chip Write Cycle

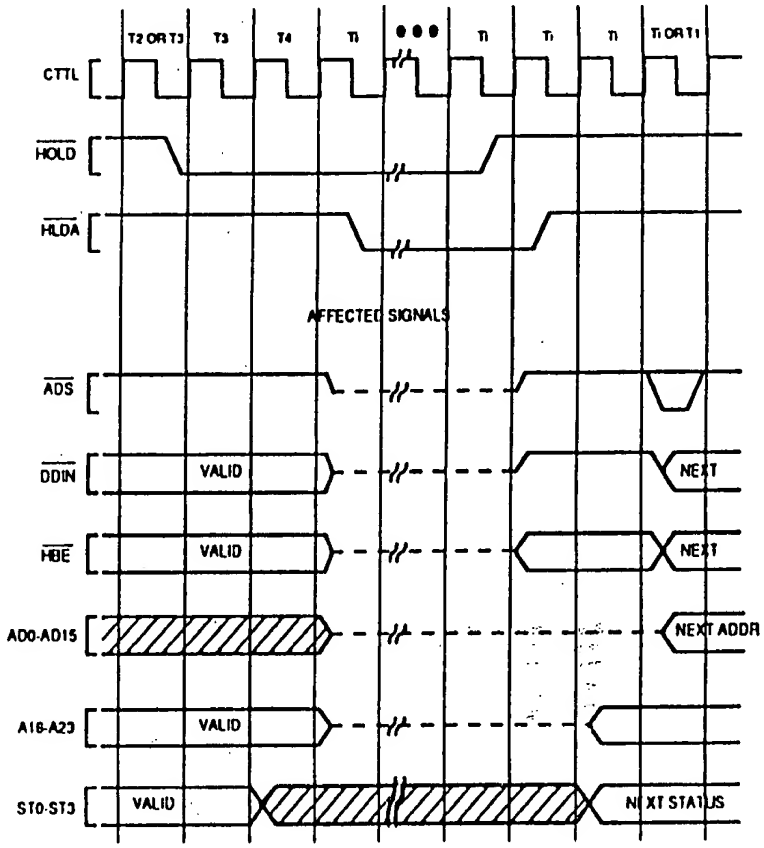


FIGURE 3-17. HOLD Timing, Bus Initially Not Idle

3.6 INSTRUCTION EXECUTION AND STATUS

In addition to the four bits of Bus Cycle status (ST0-ST3), the NS32FX16 CPU also presents Instruction Status information on three separate pins. These pins differ from ST0-ST3 in that they are synchronous to the CPU's internal instruction execution section rather than to its bus interface section.

$\overline{PFS}$ (Program Flow Status) is pulsed low as each instruction begins execution. It is intended for debugging purposes.

U/$\overline{S}$ originates from the U bit of the Processor Status Register, and indicates whether the CPU is currently running in User or Supervisor mode. Although it is not synchronous to bus cycles, there are guarantees on its validity during any given bus cycle. See the Timing Specifications in Section 4.

$\overline{ILO}$ (Interlocked Operation) is activated during an SBITI (Set Bit, Interlocked) or CBITI (Clear Bit, Interlocked) instruction. It is made available to external bus arbitration circuitry in order to allow these instructions to implement the semaphore primitive operations for multi-processor communication and resource sharing. $\overline{ILO}$ is guaranteed to be active during the operand accesses performed by the interlocked instructions.

Note: The acknowledge of $\overline{HOLD}$ is on a cycle by cycle basis. Therefore, it is possible to have $\overline{HLDA}$ active when an interlocked operation is in progress. In this case, $\overline{ILO}$ remains low and the interlocked instruction continues only after $\overline{HOLD}$ is de-asserted.

3.0 Functional Description (Continued)

(HOLD Acknowledge) pins. By asserting $\overline{HOLD}$ low, an external device requests access to the bus. On receipt of $\overline{HLDA}$ from the CPU, the device may perform bus cycles, as the CPU at this point has set AD0-AD15, A16-23 and $\overline{HBE}$ to the TRI-STATE® condition and has switched $\overline{ADS}$ and $\overline{DOIN}$ to the input mode. The CPU now monitors ADS and DOIN from the external device to generate the relevant strobe signals (i.e., $\overline{TSO}$, $\overline{DBE}$, $\overline{RD}$ or $\overline{WR}$). To return control of the bus to the CPU, the device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

How quickly the CPU releases the bus depends on whether it is idle on the bus at the time the $\overline{HOLD}$ request is made, as the CPU must always complete the current bus cycle. *Figure 3-16* shows the timing sequence when the CPU is idle. In this case, the CPU grants the bus during the immediately following clock cycle. *Figure 3-17* shows the sequence if the CPU is using the bus at the time that the $\overline{HOLD}$ request is made. If the request is made during or before the clock cycle shown (two clock cycles before T4), the CPU will release the bus during the clock cycle following T4. If the request occurs closer to T4, the CPU may already have decided to initiate another bus cycle. In that case it will not grant the bus until after the next T4 state. Note that this situation will also occur if the CPU is idle on the bus but has initiated a bus cycle internally.

Note 1: During DMA cycles the $\overline{WAIT}$1-2 signals should be kept inactive, unless they are also monitored by the DMA controller. If wait states are required, $\overline{CWAIT}$ should be used.

Note 2: The logic value of the status pins, ST0-ST3, is undefined during DMA activity.

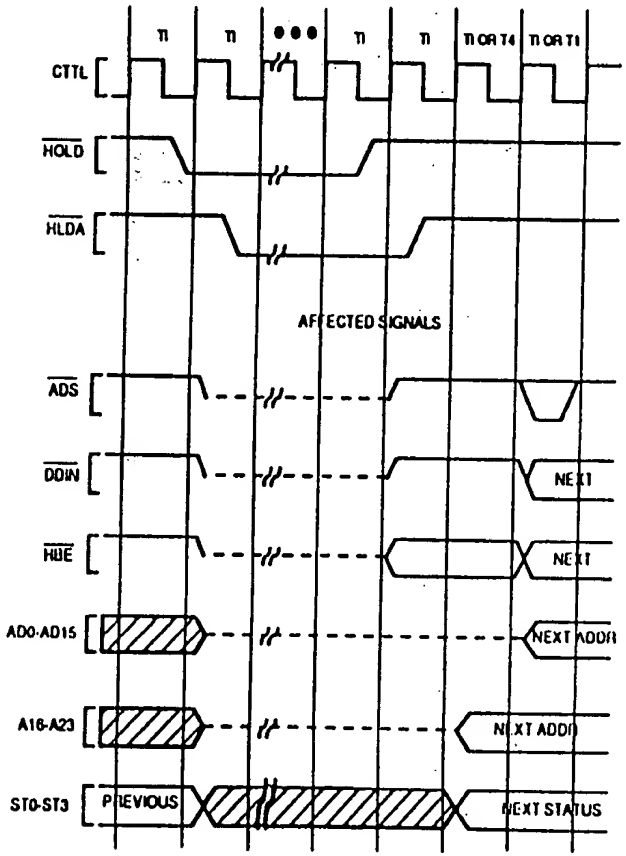


FIGURE 3-16. HOLD Timing, Bus Initially Idle

3.0 Functional Description (Continued)

3.4.8 Initiated by Off-Chip DMA Controller

Off-chip DMA Controller requests are handled by the Bus Access Control mechanism. When granted with the bus (HLDA asserted), the DMA Controller issues $\overline{ADS}$ and DOIN signals to the CPU and drives the address/data buses. The CPU supports the DMA bus cycle by generating bus control signals as $\overline{RD}$, $\overline{WR}$, $\overline{TSO}$ and $\overline{DBE}$. As a result, the DMA bus cycles are very similar to the CPU bus cycles. This simplifies the memory system design significantly.

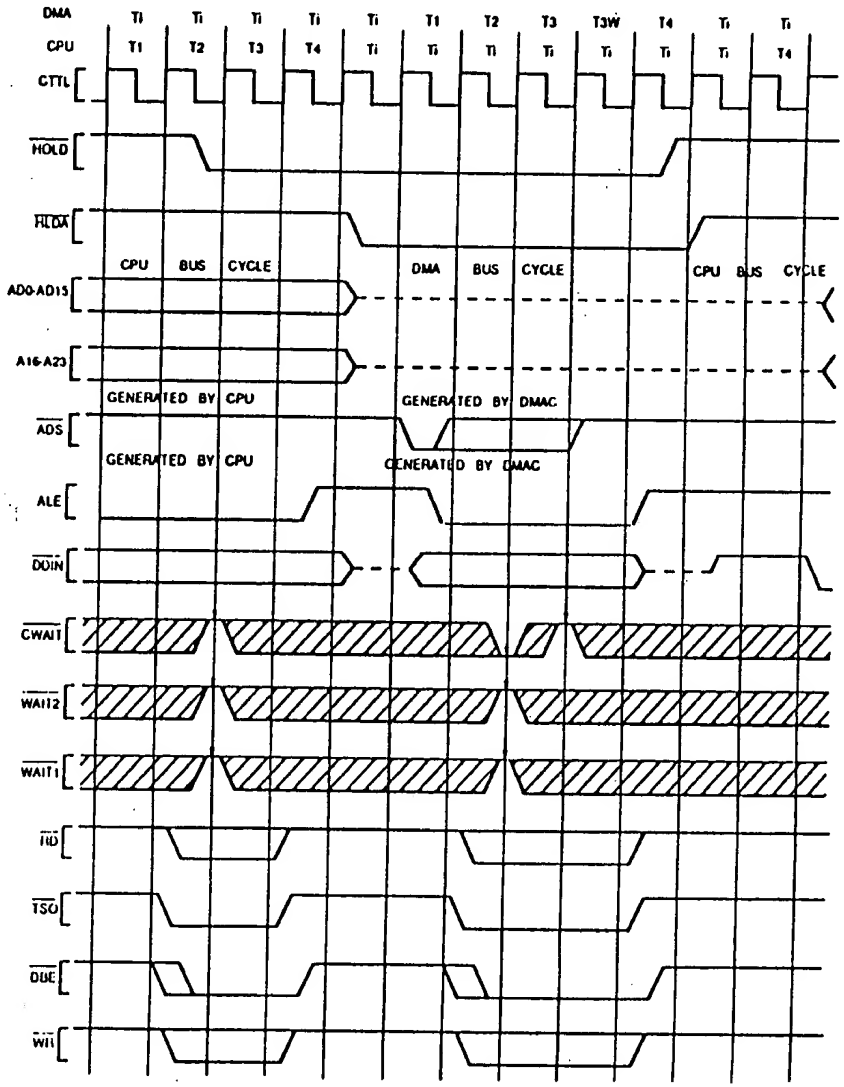


FIGURE 3-12. DMAC Initiated Bus Cycle

3.4.9 Slave Processor Communication

The $\overline{SPC}$ pin is used as the data strobe for Slave Processor transfers. In a Slave Processor bus cycle, data is transferred on the Data Bus (AD0-AD15), and the status lines ST0-ST3 are monitored by the Slave Processor in order to determine the type of transfer being performed. $\overline{SPC}$ is bidirectional, but is driven by the CPU during all Slave Processor bus cycles. See Section 3.8 for full protocol sequences.

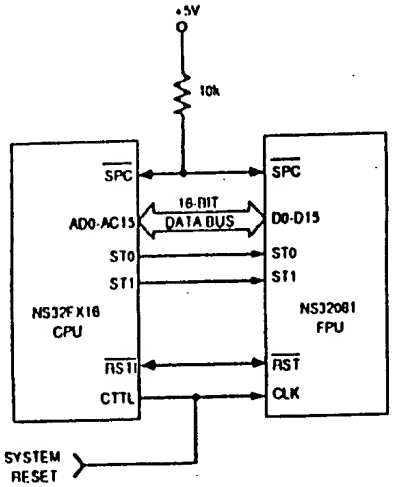


FIGURE 3-13(a). System Connection Diagram with the NS32081 FPU

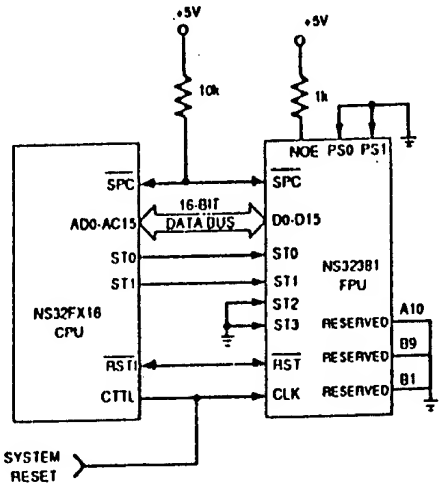
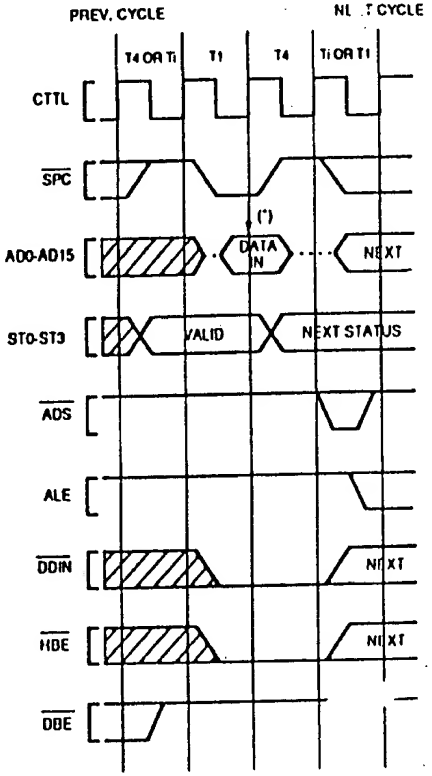


FIGURE 3-13(b). System Connection Diagram with the NS32381 FPU



(*) : CPU SAMPLES DATA BUS HERE

FIGURE 3-14. Slave Processor Read Cycle

3.4.9.1 Slave Processor Bus Cycles

A Slave Processor bus cycle always takes exactly two clock cycles, labeled T1 and T4 (see *Figures 3-14* and *3-15*). During a Read cycle $\overline{SPC}$ is active from the beginning of T1 to the beginning of T4, and the data is sampled at the end of T1. The Cycle Status pins lead the cycle by one clock period, and are sampled at the leading edge of $\overline{SPC}$. During a Write cycle, the CPU applies data and activates $\overline{SPC}$ at T1, removing $\overline{SPC}$ at T4. The Slave Processor latches status on the leading edge of $\overline{SPC}$ and latches data on the trailing edge.

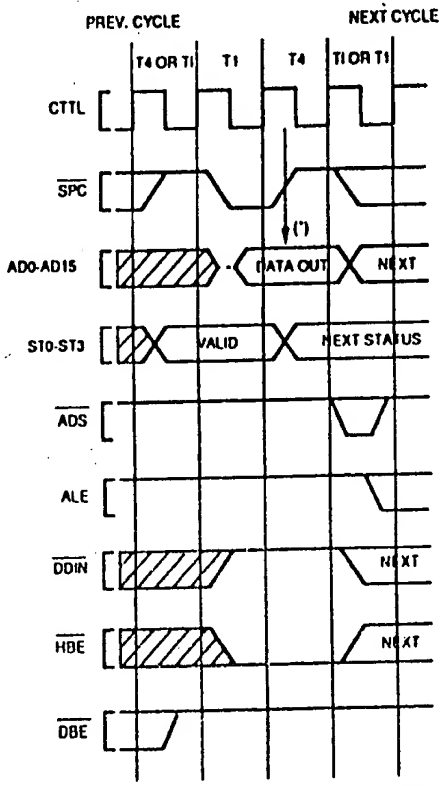
The CPU does not pulse the Address Strobe ($\overline{ADS}$) and no bus signals are generated. The ALE signal remains high during the slave cycle. The direction of a transfer is determined by the sequence ("protocol") established by the instruction under execution; but the CPU indicates the direction on the $\overline{DDIN}$ pin for hardware debugging purposes.

3.4.9.2 Slave Operand Transfer Sequences

A Slave Processor operand is transferred in one or more Slave bus cycles. A Byte operand is transferred on the least-significant byte of the Data Bus (AD0-AD7), and a Word operand is transferred on the entire bus. A Double Word is transferred in a consecutive pair of bus cycles, least-significant word first. A Quad Word is transferred in two pairs of Slave cycles, with other bus cycles possibly occurring between them. The word order is from least-significant word to most-significant.

3.5 BUS ACCESS CONTROL

The NS32FX16 CPU has the capability of relinquishing its access to the bus upon request from a DMA controller or another CPU. This capability is implemented on the $\overline{HOLD}$ (Hold Request) and $\overline{HLDA}$



(*) : SLAVE PROCESSOR SAMPLES DATA BUS HERE

FIGURE 3-15. Slave Processor Write Cycle